PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 11/30

A1

(11) International Publication Number: **WO 00/70463**

(43) International Publication Date: 23 November 2000 (23.11.00)

(21) International Application Number: PCT/US00/12724

(22) International Filing Date: 9 May 2000 (09.05.00)

(30) Priority Data:

60/134,090	14 May 1999 (14.05.99)	US
60/144,319	16 July 1999 (16.07.99)	US
09/506,024	17 February 2000 (17.02.00)	US

(71) Applicant: L-3 COMMUNICATIONS CORPORATION [US/US]; 600 Third Avenue, 34th Floor, New York, NY 10016 (US).

(72) Inventors: BUSH, Stephen, F.; 9 Sable Terrace, Latham, NY 12110 (US). BARNETT, Bruce, G.; 64 Calhoun Drive, Troy, NY 12182 (US). GALUP, Luis, E.; 153 Oak Brook Commons, Clifton Park, NY 12065 (US).

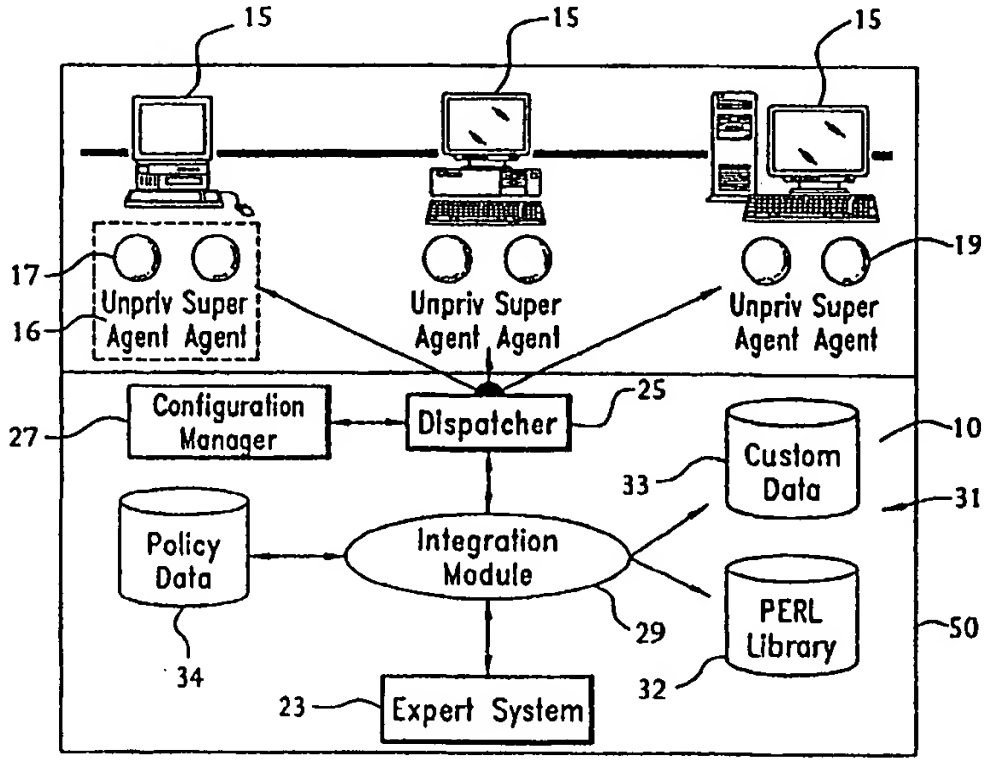
(74) Agents: ROCCI, Steven, J. et al.; Woodcock Washburn Kurtz Mackiewicz & Norris LLP, One Liberty Place – 46th Floor, Philadelphia, PA 19103 (US).

(81) Designated States: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published

With international search report.

(54) Title: APPARATUS AND METHODS FOR ANALYZING MULTIPLE NETWORK SECURITY VULNERABILITIES



(57) Abstract

A method and apparatus for analyzing multiple computer network vulnerabilities, capable of gathering vulnerability data from one or more hosts (15) within the network and generating a directed graph from the gathered vulnerability data. Nodes in the graph represent vulnerabilities within the network. Paths between nodes are edges, and represent probability values associated with moving from one vulnerability to another.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

APPARATUS AND METHODS FOR ANALYZING MULTIPLE NETWORK SECURITY VULNERABILITIES

TECHNICAL FIELD

This invention relates generally to computer data networks and, specifically, to a network vulnerability analysis tool.

BACKGROUND OF THE INVENTION

As the world transitions to more standardized digital communications driven by the various international standards organizations, and as interoperability becomes the norm for both communication networks and processing systems, information systems have and will become the targets for penetration, deception, and/or destruction by adversaries. Absent a constant vigilance and administration, any secure system will become more vulnerable over time. This is due to both internal and external sources. Internally, as access privileges are granted to new users or increased to existing users (including unauthorized access), the security profile of the system changes, usually for the worse. Client/server architectures, remote access, and trusted networks exacerbate the problem.

Externally, as the profile of the network becomes more familiar to an intruder and as penetration methods become more sophisticated, the system becomes less secure.

In the beginning of computer security, when there was little or no network connectivity, most of the attention was drawn towards insider threats and internal computer misuse. For instance, a user is granted a particular set of privileges and abuses it by accessing unauthorized files or modifying data for his or her own purpose.

During the 1980s and 1990s, as the Internet grew at an astronomical rate - from under 2,000 hosts in October 1985 to currently over 18 million hosts - firewalls drew a lot of attention. Firewalls allow companies and other entities to have partial connection to the Internet, while retaining some amount of physical isolation. Corporations were quick to create gateways from their internal networks to the Internet and used firewalls to protect their security.

Now, it appears that the focus is once again on internal threats. Many corporate intranets are now larger than the entire Internet was in the mid-1980s. With trusted hosts providing connectivity between different business units and corporate partners, companies now have a number of machines that are effectively behind the corporate firewall. Managing the complexities of connectivity between all of these hosts forces system administrators to view firewalls as only a complement to other security measures.

A system administrator has a number of powerful security and audit tools available to deal with the aforementioned problems. Many of these tools are freely available. Two major classes of system security tools that address the issue of internal threats and system integrity are 1) vulnerability assessment systems, and 2) intrusion detection systems. The former is proactive; it looks for potential system vulnerabilities. The latter is reactive; it attempts to detect that an attack or intrusion has occurred.

The present invention relates to a vulnerability assessment system, of which two types are now known to exist:

1) Single system, internal privileged assessment using software packages running on a single system with superuser privileges. One example of such a system is Computer Oracle and Password System (COPS), a UNIX security status checker which can be retrieved via anonymous File Transfer Protocol (FTP) from cert.org in -/pub/tools/cops; and 2) Distributed, external, non-privileged assessment packages that can scan several systems for weaknesses. An example of such a package is the Security Administrator Tool for Analyzing Networks (SATAN). SATAN recognizes several common networking-related security problems, and reports the problems without actually exploiting them.

In general, these packages are difficult to use, and hard to maintain. Because databases may not be updated as new information is acquired, it is difficult to keep the checks up-to-date. A person must walk from system to system with a floppy disk to check the security of each system. Even so, it is difficult to know if all of the checks are being performed properly. This is because checks are often hidden deep inside scripts that do not have any obvious relationship to the files being exercised. To learn what list of checks is being performed requires examining the source code in detail. Such an examination is burdensome and time consuming.

Another problem is the lack of a clear security policy. System Administrators typically leave the checking programs alone, or else run all of the tests, and then remove the tests that cause problems. There is no clear definition of policy for the system, and no clear relationship between the policy, the tests performed, and the tests ignored. Therefore, it is difficult to know if a system is vulnerable, and how a particular feature relates to the vulnerability.

A third flaw in the architecture of these systems is the lack of a clear hierarchy of features in each system. Currently, code is written for a single platform, and when other platforms are added, the code is modified with branch conditions for each architecture. This makes the code hard to follow, and makes it hard to know what check is being

performed.

Standard vulnerability assessment techniques are suitable for finding major security problems and may help protect a system from a brute force attack. However, they offer no defense against a hacker using a series of techniques to gain access to a system through a series of weaknesses. It is possible for a hacker to use a series of techniques to gain access to a system through a series of weaknesses of least privilege.

What is needed, then, is a vulnerability assessment tool that can diagnose and analyze multiple security vulnerabilities of a computer data network and the manner in which security safeguards strengthen those weaknesses, and can provide an easily understood display (both in 2- and 3-dimensions) of the network and its vulnerabilities.

SUMMARY OF THE INVENTION

The present invention broadly comprises apparatus and methods for analyzing multiple computer network vulnerabilities, comprising gathering vulnerability data from one or more hosts within the network; and, generating a directed graph from the gathered vulnerability data where nodes in the directed graph represent vulnerabilities within the network. Paths between nodes are defined as edges, and the edges represent probability values associated with moving from one vulnerability to another. The invention also includes two separate methods of analysis: a probabilistic approach and a maximum flow approach, both accomplished by computer software implementation of algorithms. The invention displays the results of the analysis in a number of ways, and is capable of displaying a topological display of the computer network vulnerabilities.

These and other objects, features and advantages of the invention will become readily apparent to those having ordinary skill in the art upon a reading of the detailed description of preferred embodiments and the appended claims

in view of the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a representative software architecture that can be used in association with the present invention.

Figure 2 is a representative common Application Program Interface (API) and integration module that can be used in association with the present invention.

Figure 3 illustrates a Central Assessment System (CAS) and agent dialogue in a representative secure communications protocol that can be used in association with the present invention.

Figure 4 illustrates a representative secure communications protocol that can be used in association with the present invention.

Figure 5 is a network vulnerability assessment graph.

Figure 6 is an example vulnerability graph illustrating about 2,000 vulnerabilities found on a few nodes of a network thought to be reasonably secure.

Figure 7 is an example vulnerability graph generated by the invention.

Figure 8 is a maximum flow result graph of a network.

Figure 9 illustrates data grouped by attack paths.

Figure 10 illustrates more detailed data about attack vectors.

Figure 11 quantifies the attack paths shown in Figure 4.

Figure 12 is an example of security safeguard allocation.

Figure 13 graphs the effect on vulnerability of security safe-guard allocation.

Figure 14 is an example of an attack in progress.

Figure 15 graphs the probability of monitoring an attack during its occurrence.

Figure 16 is a three-dimensional display showing the

probability of monitoring an attack during its occurrence given multiple independent detection systems.

Figure 17 shows the startup button of the software program of the invention.

Figure 18 illustrates the start-up screen of the software program of the invention.

Figure 19 illustrates the GML Open Window.

Figure 20 illustrates an example vulnerability chain.

Figure 21 illustrates a vulnerability analysis probabilistic result for the chain illustrated in Figure 19.

Figure 22 illustrates a screen capture of a vulnerability analysis graph result.

Figure 23 illustrates a vulnerability analysis maximum flow result.

Figure 24 illustrates a vulnerability analysis graph maximum flow result.

Figure 25 is an illustration of articulation points of the graph shown in Figure 24.

Figure 26 is an alternative tree down depiction of the graph node positions, showing the attacker as the root.

Figure 27 illustrates a vulnerability graph before nodes have been grouped.

Figure 28 is a screen capture similar to that shown in Figure 27, but taken after the nodes have been grouped.

Figure 29 illustrates the group control window of the invention.

Figure 30 illustrates how to add a new node to a vulnerability graph to facilitate analysis of a "what if" scenario.

Figure 31 is a screen capture of a window used to change a node label.

Figure 32 is a screen capture of a window used to modify an edge.

Figure 33 is a screen capture of the resulting graph after a node and edge have been added.

Figure 34 illustrates the textual result of a

probabilistic analysis run on the modified graph of Figure 33.

Figure 35 illustrates the graphical result of a probabilistic analysis run on the modified graph of Figure 33.

Figure 36 is a representative topological map of vulnerabilities in a network created by the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A system 100 for analyzing multiple computer network vulnerabilities in a computer network comprising a security computer and a plurality of host computers 15 is illustrated in Figure 1. According to the embodiment shown in Figure 1, a Central Assessment System (CAS) 10 resides on security computer 50. CAS 10 is in communication with a plurality of software agent sets 16. Each software agent set 16 resides on a respective host computer 15.

Each software agent set 15 includes an unprivileged agent 17 and a super agent 19. Unprivileged agent 17 is configured to communicate with CAS 10 via BSD socket connections. Super agent 19 is configured as a proxy server and as such, is accorded root privileges and network access permissions which unprivileged agent 17 does not have, in accordance with known techniques for configuring proxy servers. Typical authentication and encryption techniques are applied to communications between CAS 10 and unprivileged agent 17 and between unprivileged agent 17 and super agent 19 to prevent unauthorized access to privileged information accessible by super agent 19.

According to one embodiment of the invention, security computer 50 is adapted with a UNIX operating system and is ideally positioned in a highly secure location, such as a locked room having restricted access. The architecture of computer 50 is an automated Unix system management system, which monitors hosts connected by a network and reports proactively on system degradation and vulnerable configurations. CAS 10 resides on computer 50 and includes an

Expert System (ES) 23, a Dispatcher 25, a Configuration Manager (CM) 27, an Integration Module (IM) 29, and one or more supporting databases 31. Supporting databases 31 include PERL library 32, custom data 33 and policy data 34. Both CAS 10 and software agent set are implemented in the JAVA programming language. It should be appreciated that, although the present invention is implemented in the JAVA language in a preferred embodiment, the claims of the invention are not so limited, in that the invention could be readily implemented in other languages by one having ordinary skill in the art.

ES 23 comprises a plurality of reasoning modules which implement rules upon which diagnoses are based. In addition, ES 23 contains status and configuration data about networked UNIX nodes in order to diagnose vulnerable systems. The software agents 16 provide data about different hosts 15 and accounts. The ES 23 may ask the agents 16 for information about each host 15 or account. The Perl Library 22 contains a collection of methods that can be dynamically downloaded to the software agent. This allows the rules in the ES 23 to be reused, as it may have rules and understand the class or type of object, as well as the methods used to obtain information from the objects. It can know how some values returned from these queries indicate particular symptoms of system problems. Based on this information, the ES 23 may decide to query more information, initiate a vulnerability in the database, or take corrective action.

The Dispatcher 25 functions as a message switchboard between the ES 23 and software agents 16. The CM 22 provides basic configuration information about the network and hosts 15. The Custom Database 33 contains class instances of the network (users, hosts, directories, and vulnerabilities). The Policy database 34 is discussed *infra*.

The IM 29 is the mechanism by which the ES 23 and the software agents 16 access system and network information. It also is the interface for the system to other security applications. The outputs of various security tools are instantiated into common objects via the IM 29.

The software architecture described above was developed based on the premise that the degree of protection from information warfare attacks is highly dependent on the amount of time and effort invested in building and maintaining system security defenses. The solution offered by the present invention is to off load the work and tasks of network management to software agents. Each agent 16 operates independently, but they all cooperate in monitoring the system. Collectively, the software agents 16 achieve the overall goal of system monitoring and intrusion detection. This approach provides significant advantages in terms of scalability, flexibility, and efficiency.

The software agents 16 are TCP-based server programs started up by the root user. Written entirely in Perl 5 and less than 50KB in size, the software agents 16 are designed to have minimal effect on host performance. The agent code does not reside on any network disk. This prevents an intruder from modifying the agent code to affect agent behavior.

The software agents 16 are dynamically extensible because methods and instructions can be downloaded to them from the CAS 10. Because the software agents 16 do not contain a collection of security checks and information-gathering modules embedded in them, this greatly helps reduce the amount of space the agents occupy in memory. The agent's actions are not scripted; the CAS 10 can dynamically choose which actions to invoke, and in what sequence, in response to the state of its external environment.

The CAS 10 communicates to the unprivileged agent 17 by sending it a method and the reference to an object to which the method is to be applied. The object is referenced by the object type, followed by the name of that object. For example, to reference the route account on machine tango, the object reference would be account/tango/root. Basic uploadable methods include:

scan_all_patches - given a reference to a host object, an agent will scan for the correct installation of security patches for a particular operating system. This is

accomplished by taking the MD5 signature of every file associated with a particular patch and comparing it to the signature of the file currently on the system. Operating systems that are currently supported include SunOS 4.1.3, Sun OS 4.1.3 Ul, Solaris 5.2, Solaris 5.3, Solaris 5.4, and Solaris 5.5.1. The patch information is updated regularly by the CAS via ftp.uu.net (the official distribution for SUN security patches).

scan_trojan_directories - given a reference to particular directory or user search path, the agent will scan for trojan horse vulnerability. This is accomplished by recursing through every file and directory to determine which files are group or word writable. The method can also handle soft and hard links.

read_all_data - this is a generic method that allows an agent to read instantiated vulnerabilities from the custom database. This will basically allow different security applications to communicate to the agents. The Integration Module takes the input for any arbitrary number of security applications and instantiates vulnerabilities into the common OO database as shown in Figure 2.

An IDS can generate vectors and the IM can instantiate vulnerabilities into the common OO database. A security agent reads the database to gather security information. Other information reporting methods include:

report_all_vulnerabilities -list all vulnerabilities on a particular machine

dump_root_vulnerabilities - list all root vulnerabilities on a particular machine

object_count - the number of objects the agents knows about from the custom database

The count of the number of vulnerability instances is a primitive measure for system administrators to show progress in securing their networks. The present invention as described *infra* provides a closer examination of each vulnerability listed to clearly indicate what steps are

necessary to reduce the number of instantiated vulnerability objects.

The interface between the privileged agent and non-privileged agent must be secure. An intruder who has access to the interface cannot ask the system to weaken the system security. The interface is designed to be as simple as possible, as complex systems are harder to secure.

Consider an example where the CAS asks the unprivileged agent on a host to gather some information. The unprivileged agent wants to examine the security of a file, and does not have permission to do so. It asks the privileged agent to get the information, and pass the results back to the non-privileged agent. This dialogue is shown in Figure 3.

The format of each request is a packet containing the following four fields: type, length, data, and signature. The type field identifies the type of packet, the type of signatures used, and the format of the packet. Some packets could contain binary information, others ASCII. The length indicates the amount of data (and/or the length of the signature field). The data is the command issued. It is always encrypted using DES-ECB. The signature can vary with packet type (some types may not have a signature field). One of the common signatures will be the MD5 hash of the data, encrypted using the secret key shared between the two systems. In this manner, the integrity of the command, and authentication, can be confirmed before the command is executed.

For each communication session between either the CAS and the unprivileged agent or between the unprivileged agent and the super agent, there are authentication mechanisms and an exchange of keys. This security protocol is shown in Figure 4.

There is a secret master key and secret master TCP port that the two entities use to establish a secure communications channel. The shared key and port is known only between the entity pair. The master key and a master TCP port are used to establish a session key and session port. At each intermediate step, a random challenge is issued to

authenticate the sender and to prevent message replay. Once the secure communication channel is established, the two entities can continue to communicate, and no additional handshaking is necessary. Table 1 below shows the contents of the payload, the encryption key used, and the TCP port used at each step of the protocol.

Table 1. Payload, Key, and Port for Protocol

Stage	Payload	Encryption Key	TCP Port
1	Request for Conversation	Master	Master
2	Random Challenge	Master	Master
3	Random Challenge and Session Key	Master	Master
4	Random Challenge and Session Port	Session	Master
5	Random Challenge and Information Request	Session	Session
6	Random Challenge and Information	Session	Session

The privileged agent has a database and caching

mechanism that identifies the list of commands, and the security requirements necessary for each command. For instance, any system might be able to see if the privileged agent is running, and obtain the revision of the agent. Or the agent may refuse to acknowledge an outside query. The choice is up to the agent, and the person who installed the agent.

It is unlikely that a complete list of security problems exists. New ones are always being reported. The architecture is not intended to be a list of unrelated problems, but an organization based on operating system, release level, and policies. The goal is to have an integrated knowledge base, so that an intelligent system can react to exploitation problems without disabling the entire network. The Security Policy Database (SPD) provides a structured, quantifiable way to record information about systems and their flaws as they are discovered.

There is a direct relationship between the object classes and the information in the database. This object model allows evolution and code reusability, allowing modification of classes with minimum impact. Five kinds of items in the SPD are as follows:

Objects: Objects are resources and actors on a computer system. They may be such concrete or abstract things as files, peripherals, network ports, and users. Objects are represented by a list of labeled data values.

Systems: Systems are the active software systems providing services on a network. Examples of systems are the kernel, mail and Web servers, and the file system.

Abilities: Abilities are the actual services available from systems that act upon or at the request of objects. Abilities are represented by a set of preconditions indicating states of objects necessary to use the ability, and a set of post-conditions indicating changes to the objects from the use of the ability. Abilities may be intentional, such as the ability for the owner of a file to modify it. They may also be unintentional, such as the ability of network user to gain root access from certain versions of the mail system.

Policy: Policy describes what is allowed and disallowed (*e.g.*, world readable files, no passwords in the clear over the wire, authenticating based on IP address, File Sharing, unauthenticated mail, set-uid files on home directories, finger, "." in a search path, *etc.*). The policy has impact levels. That is, each policy should indicate the importance of the policy. A simple rating system may include states: (Allow, Should, Should Not, Disallowed, Unknown). A policy may start out as "Unknown/Don't Care", then change to "Should Not", and later change to "Disallowed" - based on conditions. Other schemes may also be appropriate. One option is to have a variable policy, with a range of values, ensuring genetic variability in systems.

Regions: Regions represent the organization of processes, corresponding to a chain of command. A relationship between bosses and workers is defined by this object class, and its relation to other objects of the same class. One model may be simple: Each region has one boss, and each regional boss may have several sub regions reporting to it. Other models may include two levels of command for each region, or shared authority. Regions group systems, and may be geographical, or authoritative.

Figure 5 is an instance of a typical security object model. Just as a virus uses host cells to reproduce, an attacker enters the network and can choose from a variety of vulnerabilities to take control of the network. In Figure 5, an attacker decrypts the password of User 2 on Host 1. The cost to the attacker is (10,0,0,0) which is a vector of the form: (password decryption, NFS spoofing, host spoofing, application fault). From that point, an attacker may illegally modify File System 1 on Host 1 with a cost of (0, 10, 0, 5), or decrypt the password for User 1 on Host 1 with a cost of (10, 0, 0, 0). Host 3 in Figure 5 is an example of host spoofing in which the attacker can use File System 1 on Host 1 in order to change the identity of a host. From the graph shown in Figure 5, it should be understood that the network

is as vulnerable as the weakest path.

The preceding description is intended to set the background, by way of example, for a representative object model, software architecture, software agents and secure communications protocol that may be used in conjunction with the present invention. It should be understood, however, that other models, architectures, types of software agents and communication protocols may be used, and thus the preceding description is not necessary to enable one having ordinary skill in the art to make and use the invention, which is described here below.

In the description that follows, the method and apparatus of the invention will be referred to as the "tool" for convenience. The tool is a Java-based network vulnerability and analysis tool.

Figure 6 displays 2000 vulnerabilities found on a few nodes of a network that were thought to be reasonably secure. Vulnerabilities are displayed in Figure 6 by host and type. The number along each edge of the graph represents the number of opportunities available to the attacker to reach the next vulnerability. Using this information, the tool has several algorithms for determining the vulnerability, V.

In a preferred embodiment, the tool uses two fundamental techniques for determining vulnerability to attack. Both techniques are based on determining "insecurity flow." The tool can display the results at various levels of detail, including the individual host level, vulnerability types, host types, or individual vulnerabilities.

The tool can automatically generate a directed graph representing the security vulnerabilities of a network. This information is gathered from the network security software agents described above. The security vulnerability graph for a typical network can be extremely dense; however, the object-oriented nature of the security model described above is useful in choosing the level of abstraction required. For example, it may be possible to display the vulnerability graph for Unix hosts in general and hide the details of individual

Unix variants. The tool determines the degree to which specified targets within the network can be compromised. The vulnerability chain is displayed as a directed graph. Nodes represent vulnerabilities whose security may be compromised and edges represent paths from one vulnerability to another. The larger the value of the edge label, the greater the vulnerability. Figure 7 shows an example vulnerability graph generated by the tool.

One of the security assessment operations the tool can perform is to determine the vulnerability of a particular entity given an attack on a particular node. The target entity Host C Vulnerability 4 is identified by a white cross-hair in Figure 8 and the attacking node is labeled Attacker with the flow identified by the label of its connecting path. The optimal vulnerability path is the sum of flows into node Host C Vulnerability 4 as shown in Figure 8, a flow strength of 6.0.

In Figure 8, the optimal path that the attacker can take to reach the target is shown. Thus the tool provides the ability to examine how the placement of security safeguards such as intrusion detectors within the network affect total network security. In effect, this tool becomes a security modeling tool, where one can experiment with the placement of security safeguards representing such entities as firewalls, intrusion detectors, and access lists. These can be positioned at various locations in order to determine network security.

The tool allows various types of node groupings in order to help visualize the vulnerability paths. In Figure 9, all object types are grouped together. The nodes could also be grouped by such characteristics as hostname or subnetwork. In Figure 10, the vulnerabilities which have been identified and grouped as vectors to vulnerability targets have been expanded to show more detail about the individual vulnerabilities. In Figure 11, all 40 parent objects of sun4/bin are grouped within a single node. Also note that the root account is clearly visible as reachable through the vulnerability path.

It becomes clear that defensive security safeguards cannot be studied independently of offensive information warfare. Thus, a tool that can accurately study both is desirable. Initially, perfect information is assumed to be available to both the attacker and defender. Later, the effects of the more realistic case of imperfect information is considered, since neither the attacker nor the defender can have complete knowledge of one another's state.

The attacker can use one or more combinations of the following types of attack. An attack consists of a string of one or more of the following classes:

interruption - Interruption is the termination of a service required by the network. Purposely overwhelming an application so that it cannot service other users is an example of interruption.

interception - Interception is obtaining information from the network useful for an attack. An example of interception is obtaining information from the network useful for an attack.

modification - Modification is a change of information in the network which weakens network security. Planting a virus is an example of modification.

fabrication - Fabrication is the construction of data for the purpose of weakening network security. This could include guessing passwords or building and sending invalid protocol data units.

Barriers exist to these forms of attack besides firewalls as shown in Table 2. Note that these defenses are effective independent of time. The simplified attack cost vector is shown in Equation (1). In this analysis vulnerability is quantified in units of time. For example, fabricating a password on a particular node will cost an attacker the amount of time which depends on the rate that new passwords can be generated, the number of accounts on the target node, and how well the passwords have been chosen.

TABLE 2

TIME INDEPENDENT (NON-POLLED) DEFENSES

Attack	Defense	Variable Name
interruption	improved design	id
interception	encryption authentication non-repudiation	en au nr
modification	signature	si
fabrication	signature	si

$$cf(p) = \begin{array}{l} \\ \text{interruption} \\ \text{interception} \\ \text{modification} \\ \text{fabrication} \end{array} \begin{pmatrix} cf_i(p_i) \\ cf_1(p) \\ cf_2(p) \\ cf_3(p) \\ cf_4(p) \end{pmatrix} \qquad (1)$$

Using the tool and analysis methods of the present invention, a network security analyst can allocate security safe-guards in order to minimize the entire network vulnerability, or to minimize the vulnerability from known attack points to particular targets.

As an attack takes place, the defender can use the tool to study the effectiveness of various strategies using actual network vulnerabilities, but within the safety of a simulation environment. The analysis tool can be used to determine the optimal location of services to be cut. The effect of concentrating on reducing specific vulnerability classes will be the focus, rather than cutting-off access to

entire network hosts that have been compromised. Also, by studying the past history of an attack, it will become apparent which vulnerability classes a particular attacker prefers to exploit.

From a fundamental network vulnerability flow viewpoint, the strategy of allocating safe-guards in combinations of serial and parallel strategies can be examined. Figure 12 shows Network Insecurity Path Assessment Tool analyzing an attack from host A to host B. In this case, the number of opportunities have been normalized into probabilities. Figure 13 shows the results as security safe-guards are removed. The solid line is the vulnerability of a single connection from the attacker to the defender having the same vulnerability flow as the links shown in Figure 12. Below a probability of 0.6 the diversity of vulnerability types helps to increase security, but interestingly, above 0.6 it does not.

Once an attack has been detected, the network command and control center can respond to the attack by repositioning security safe-guards and by modifying services used by the attacker. However, cutting-off services to the attacker also impacts legitimate network users and a careful balance must be maintained between minimizing the threat from the attack and maximizing services to customers. For example, various stages of an attack are shown in Figure 14. Since the allocation of security resources never changes throughout the attack, the vulnerability of the target increases significantly with each step of the attack.

Because vulnerabilities change over time, the network monitoring tool described quantifies the vulnerability of a system in terms of percent of patches which fail to have the correct signature (p_f), percent of files which are accessible to others besides the owner (p_a), and percent of passwords which can be guessed with a given password generation tool (p_g). Clearly, vulnerability checks such as these increase the security of the network. The effectiveness of a network monitoring strategy is quantified by both the

type of information gathered and the frequency that the information is updated. If the information is not updated frequently enough, an attacker may have penetrated network security and left before network security is aware of the situation.

In this analysis, a path with perfect security has a $cf_i(p_i) \dashrightarrow \infty$, and a path with no security has a $cf_i(p_i) \dashrightarrow 0$. The vulnerability of a path is defined as the inverse of the $cf_i(p_i)$. An estimate of the effectiveness of the monitoring system is based on a profile of network security attacks on the Internet and the following parameters: time to monitor patches, Trojan horses, passwords, and any other vulnerabilities (t_s), the attack rate is Poisson and the attack duration is exponential with average a_r, and monitoring is performed every λ_m seconds. The average attack rate, based on Internet incident reports from an anonymous site for a six year period, is five attacks per month. Also, the Defense Information Systems Agency has determined by experimental means that only 0.7% of incidents are actually reported. Thus the probability of detecting an attack while the attack is taking place along path i is shown in Equation (2), and the results are graphed in Figure 15 with $a_r = 5/(0.007)(30)(24)$, the y-axis is P[*detect*], and the x-axis is $\lambda_m + t_s$.

$$P_i[\text{detect}] = 1 - e^{-a_r \frac{1}{\lambda_m + t_s}} \qquad (2)$$

Thus, for each path in the network security vulnerability chain, the cost to the attacker is the probability of being detected multiplied by the cost function that the additional monitoring provides. Thus the total cost function is shown in Equation (3).

$$cf_i(p_i) = \left(1 - \prod_{n=0}^{i} \overline{P_n[detect]}\right) (p_g + p_o + p_f) + (id + en + au + si) \qquad (3)$$

Figure 16 shows the increase in probability of detection as an intruder passes through multiple systems where each system has its own independent detection system.

The tool has served as experimental validation of a variety of techniques to analyze a communications network for vulnerabilities. It can be taken a step farther, however, by adding the capability of automatically determining the placement of security safeguards based on predetermined cost limitations.

Let S be the placement of security safeguards (*e.g.*, encryption, firewall, authentication), and V be the vulnerability. Let C be the cost of the safeguards, and L be some threshold to the acceptable cost. The Object Function (4) sets up the optimization problem:

$$\min V(S) \quad (4)$$
$$\text{subject to } C(S) < L.$$

Network security tends to hamper the effectiveness of the network to legitimate users. Taking this into account, let CS represent the network service to legitimate users of the network, with a minimum accepted quality, Q, and V(A) be the vulnerability of the network to a particular attacker, A. The Object Function (5) sets up the optimization problem:

$$\min V(S,A)$$
$$\text{subject to } CS > Q \quad (5)$$
$$C(S) < L.$$

The tool is written in Sun Java with JDK 1.2. Secanal is the main Java application. The use of the tool will now be explained in detail.

Enter "java Secanal" to begin the application. A large button should appear as shown in Figure 17. The complete command line arguments are "java Secanal [-b filename][-f filename][-s nodenumber][-l layout]". The -b option loads an

IW file explained *infra*, the -f. option loads a GML file explained *infra*, the -s option begins the applications with a given node already selected, and the -l option is the label of a menu item to be executed upon start-up.

Click on Start Security Analysis to bring up the screen shown in Figure 18. Note that at any time nodes and edges can be added, deleted, or modified in a vulnerability graph in order to test "what if" scenarios. The toggle switches in the upper left allow Nodes and Edges to be created or selected. Clicking on the graph in the mode shown in Figure 18 will create a new Node.

The graph area is three-dimensional; the view angle can be changed by clicking within the viewing angle area in the lower left of the screen. An "x" will be placed in the view area indicating the corresponding angle of θ and ϕ. In addition the scale can be changed. The viewing offset is required because the graph area is larger than what is displayed within the window. Click on File from the menu along the top of the screen. The window shown in Figure 19 will appear. Open the gmi directory and open example.gml. GML is a standard notation for representing graphs and can be read by other graph applications such as graphlet (http://www.uni-passau.de/graphlet/).

Select example.gml and the example graph should be loaded as shown in Figure 20. The nodes represent vulnerability classes, the edges represent the number of opportunities to advance from one vulnerability class to another.

There are two main algorithms that can be run; the first is a probabilistic analysis and the second is a maximum flow analysis. First, the probabilistic analysis will be discussed. Select a node to be the target of the attacked by clicking on the Select Nodes toggle button. Then select a node (*e.g.*, hosts C Vuln 4). A white cross hair should appear over the node to indicate it has been selected. Choose Algorithms; choose Security Analysis Models and finally choose probabilistic analysis. A text window shown in Figure 21

should appear which states the probability of successful attack followed by the result graph shown in Figure 22. The result graph shows the most probable path of attack highlighted. The edge values are normalized between 0 and 1 to represent the probability of an attacker choosing that path.

The analysis can also be run using the maximal flow algorithm as follows: Choose File and Open GML. Then choose the gml directory and choose the example.gml file. The graph window should appear as shown in Figure 20. Select Hosts C Vuln 4 again and choose Algorithms Security Analysis Models and Max Flow Analysis. The text window shown in Figure 23 should appear as well as the graph results shown in Figure 24. The edge values have been changed to show the maximum flow along each edge towards a target node. In this case there is a flow of 1.0 and a flow of 5.0 which can reach the target node. Now choose File and Exit This Window in order to close the window in Figure 24. The original window should still be open.

One method of adding security may be to partition the vulnerabilities so that paths do not exist across the vulnerability chain. One method to determine how to partition the chain is to determine the articulation points. These are single nodes which if removed from the graph will partition the graph into multiple disconnected subgraphs. In order for this to work the graph must be undirected. Choose Properties and Directed. This will toggle the graph in undirected mode; the arrows will disappear from all the edges. Next choose algorithms and Biconnectivities and choose Find Articulation Points. A window as shown in Figure 25 will appear. In order to identify the node number on the graph, select a node and the node number and position will appear along the top of the main window.

The tool is also capable of executing other commands and controls such as automated node layout, importing and exporting vulnerability graphs, filtering, grouping, and automatically adding an attack node.

The options labeled Tree and Spring under Algorithms are different methods for displaying the graph node positions. For example, select a node to be the root of the tree and then select Tree and Tree Down. Figure 26 shows the result of selecting the attacker as the root in choosing Tree Down. The Spring button attempts to layout the nodes such that edges modeled as springs and the nodes are positioned such that the energy between the springs is minimized.

An optional argument to Secanal is -b filename.filename is the complete path and file in which data collected from the IW agents are collected. The file format is similar to the following:

```
Start up server on port 1661
Server listening on port 0 using fileno 5
    Server.pl: Boss send us message of: read_all_data
host/HostA
IW.pl:handle_request
read relationship Relationships
read os PERM.Solaris.2.5.1-SunOS/5.5.1
Found OS SunOS/5.5.1
IW.pl.handle_request:exit
Server.pl: Boss send us message of: report_all_files
host/HostA
IW.pl:handle_request
ID:Vulnerability/*/991
NAME:Vulnerability/hostb/14
          data[count]=0
          data[name]=Vulnerability/hostb/14
          data[type]=Trojan
          data[vector]=Directory=HASH(0x51ca54)
          data[victim]=Account=HASH(0x460fc8)
          data[attacker]=Account=HASH(0x46de24)
          Reference(Account)=(account/hostb/root)
```

When importing vulnerabilities, there can exist many thousands of fundamental vulnerabilities which can overwhelm this tool and the user if they are displayed individually.

However, vulnerabilities can be grouped based on Host, Id, Type, and a combination of Host and Type. This means that only one vulnerability will appear for each host, vulnerability identifier, vulnerability type, or a combination of host and type. Choose Algorithms and Filter Nodes and then one of the above filter types. Once this has been done, the tool will remember that filter and apply it each time vulnerability data is imported. To import vulnerability data, choose Algorithms and Import and Update Security Model. Note that imported vulnerability data updates the graph. For example, if a vulnerability graph is already displayed on the screen then importing vulnerability data will add the data to the graph.

There are two methods for exporting vulnerability graph data. The first is to save the graph in a GML file. This can be done choosing File and Save As GML. The second export format is a format readable by Mathematica. Choose Algorithms and Export Data and Convert to Mma. The data will be saved in a directory named mma assumed to exist under the current directory in which Secanal resides. The files are Adj.mma (Adjacency Matrix), Arc.mma (Edge Index Matrix), Cons.mma (Constraint Matrix), Flow Adj.mma (Adjacency Matrix with Flow values), and Inc.mma (Incidence Matrix). In addition to the above files, Adj.mo and FlowAdj.mo are generated in the mma directory. These files contain the data only arranged in row-column format.

Vulnerability nodes can be grouped together in a single node as follows. Choose Algorithms and Group Nodes and then choose one of Host, Id., Type, Host and Type, or common Child. In Figure 27, a vulnerability graph is shown before nodes have been grouped. After choosing Group and Host, the graph in Figure 28 is created. Choosing Edit and Group Control-seeNode brings up the menu shown in Figure 29 which can be used to create or remove selected groups.

In order to facilitate adding an attack node, select the node to which the attacker should be adjacent and then choose Algorithms and Security Analysis Models and Add Attack Point. An attack node will be automatically created and

attached directly to the selected node. Note that the attacking node must be labeled "Attacker" in order to be recognized as the attacker in the algorithms.

The tool is also capable of manually editing vulnerability graphs in order to analyze "what if" scenarios.

As an example, load the example gml graph as explained *supra*. Choose the Select Nodes button. The goal of this example will be to move the attack node from Host A Vulnerability 1 to Host B Vulnerability 2. This will demonstrate how to delete and create nodes and create and modify edges. Select the Attack and then choose Edit and Delete Selected Items. Both the Attack node and its adjacent edge will be removed. Select Create Nodes. Click on the graph near Host B vulnerability 2. This will create a new node as shown in Figure 30.

Choose Select Nodes and double-click on the new node. The window shown in Figure 31 should appear. Enter Attacker for the label then choose apply. The label should be changed from a node number to "Attacker."

Next choose Create Edges and select the Attacker node. The new edge should follow the cursor as it moves. Select Host B Vulnerability 2. An arrow should appear connecting the two nodes and directed from the Attacker to the vulnerability node. Choose Select Edges and double-click on the new edge. The window shown in Figure 32 should appear. Enter a value for the label and choose Accept. The graph should appear as shown in Figure 33.

Select Host C Vuln 4 and run the probabilistic analysis. The textural result is shown in Figure 34 and the graphical result is shown in Figure 35. Notice that the probability of successful attack is 0.226 in this analysis and 0.729 in the previous case. This should be expected since the path of most probable attack is longer in this analysis.

The present invention is capable of displaying the results of the vulnerability analysis in a number of ways. As seen above, the results can be displayed either graphically or texturally. In addition to the 2-dimensional graphical

analysis, the invention provides a 3-dimensional topological display of the vulnerabilities in the network. This technique allows easy identification of the security weaknesses within the network by displaying a clear 3-dimensional view of mountain peaks and valleys super-imposed upon the network node layout. The node layout can be grouped by physical location or by type. The invention then allows one to examine the impact upon the entire network of the location of various security safeguards within the network, thus facilitating a security cost-benefit analysis and optimizing the placement of security safeguards. Because it is difficult to know from where an attack will take place, the flow from every node to every other node is calculated. A contour is drawn based on the accumulated vulnerability and the distance from each node. An example contour is shown in Figure 36. The contour shows the rate of change of vulnerability as one moves through the network. The resulting topological map or an alternative density plot graph provides quick visual information indicating where vulnerabilities lie. This topological display aspect of the invention was written and implemented in *Mathematica*.

The overall vulnerability of network is represented by a directed graph of all vulnerability chains or paths that an attacker could use to invade the network. The present invention allows easy identification of the security weaknesses of the entire network to specific threats by identifying the path of least resistance to the attacker's target. The invention then allows one to examine the impact of locating various security safeguards within the network, thus facilitating a security cost-benefit analysis and optimizing the placement of security safeguards. Finally, one embodiment of the present invention displays the results of the analysis in both a 2-dimensional and 3-dimensional (topological) display.

While only certain preferred features of the invention have been illustrated and described, many modifications and changes will occur to those skilled in the

art. It is, therefore, to be understood that the appended claims are intended to cover all such modifications and changes as fall within the true spirit of the invention.

We Claim:

1. A method of analyzing multiple computer network vulnerabilities, comprising:

gathering vulnerability data from one or more hosts within the network; and

generating a directed graph from the gathered vulnerability data where nodes in the directed graph represent vulnerabilities within the network, and paths between nodes are defined as edges, and the edges represent probability values associated with moving from one vulnerability to another.

2. The method of claim 1, wherein all edges between the same two nodes are normalized into a single probability value.

3. The method of claim 2 further comprising:

multiplying probabilities of paths in series from a source node to a destination node;

adding probabilities of paths in parallel from the source node to the destination node; and

determining from the multiplication and addition the probabilities of all routes from the source node to the destination node.

4. The method of claim 3, wherein a most probable route between the source node and the destination node represents a most vulnerable attack path.

5. The method of claim 2, further comprising:

using an optimization technique to determine maximum insecurity flow from a source node to a destination node, wherein the optimization technique maximizes flow from the source node while conserving flow through intermediate edges of the graph.

6. The method of claim 2, further comprising:

finding a minimum number of nodes that can be

deleted from the graph resulting in a partitioning of the graph into at least two isolated sections.

7. The method of claim 2, further comprising:
displaying the directed graph.

8. The method of claim 7, wherein the display comprises a two-dimensional graph displayed on a monitor.

9. The method of claim 7 wherein the display comprises textural information displayed on a monitor.

10. The method of claim 2 further comprising:
multiplying probabilities of paths in series from each node in the network to every other node in the network;
adding probabilities of paths in parallel from each node in the network to every other node in the network; and
determining from the multiplication and addition steps the probabilities of all routes from each node in the network to every other node in the network.

11. The method of claim 10, further comprising:
generating a topological display of the directed graph.

12. The method of claim 2, further comprising:
using an optimization technique to determine maximum insecurity flow from each node in the network to every other node in the network, wherein the optimization technique maximizes flow from each node while conserving flow through intermediate edges of the graph.

13. The method of claim 1, further comprising:
determining a placement of security safeguards based on predetermined cost limitations.

14. The method of claim 13, further comprising:

determining the placement of security safeguards based on a minimum accepted quality of network service to legitimate users of the network.

15. A system for analyzing multiple computer network vulnerabilities, comprising:

a processor that includes computer-executable instructions for gathering vulnerability data from one or more hosts within the network; and generating a directed graph from the gathered vulnerability data; wherein nodes in the directed graph represent vulnerabilities within the network, and paths between nodes are defined as edges; and wherein the edges represent probability values associated with moving from one vulnerability to another.

16. The system of claim 15, wherein the processor further includes computer-executable instructions for multiplying probabilities of paths in series from a source node to a destination node; adding probabilities of paths in parallel from the source node to the destination node; and determining from the multiplication and addition the probabilities of all routes from the source node to the destination node.

17. The system of claim 15, further comprising:

a monitor for displaying the directed graph.

18. A computer readable storage medium, comprising computer-executable instructions for:

gathering vulnerability data from one or more hosts within a computer network; and

generating a directed graph from the gathered vulnerability data, wherein nodes in the directed graph represent vulnerabilities within the network, and paths between nodes are defined as edges, and wherein the edges represent probability values associated with moving from one vulnerability to another.

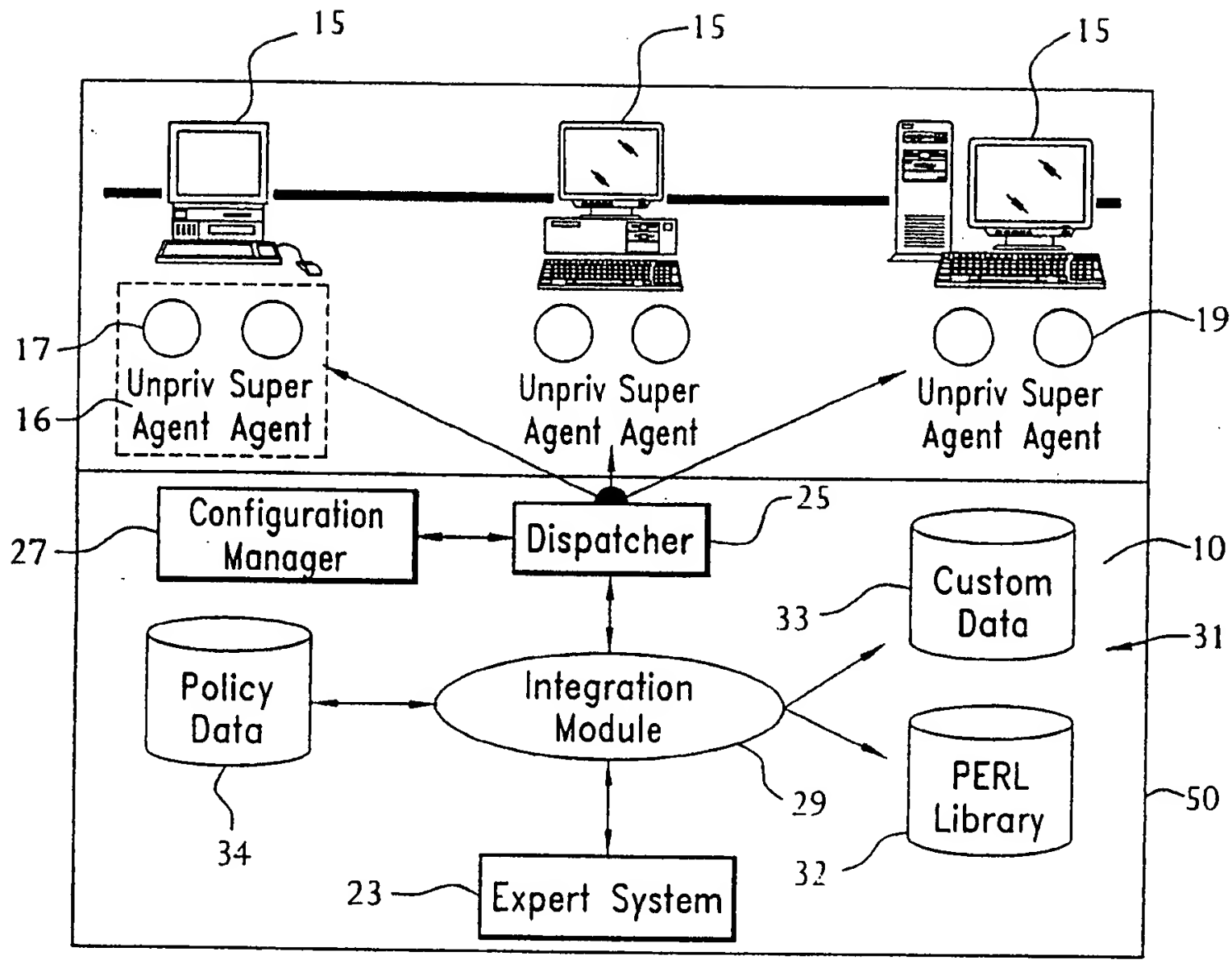

100
15
15
15
17
16
Unpriv Super Agent Agent
Unpriv Super Agent Agent
Unpriv Super Agent Agent
19
Configuration Manager
27
Dispatcher
25
10
Custom Data
33
31
Policy Data
34
Integration Module
29
PERL Library
32
50
23
Expert System

FIG. 1

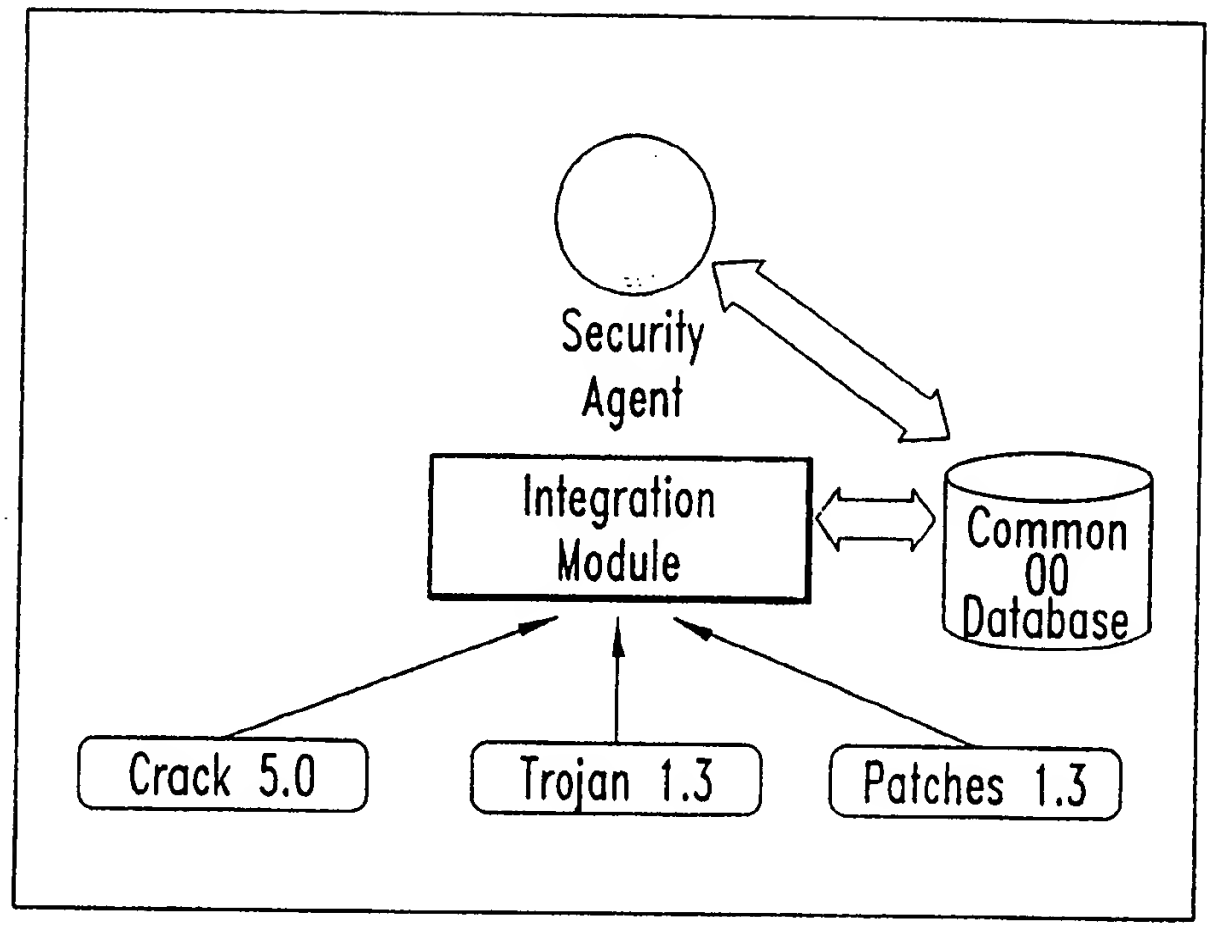

Security
Agent
Integration
Module
Common
00
Database
Crack 5.0
Trojan 1.3
Patches 1.3

FIG. 2

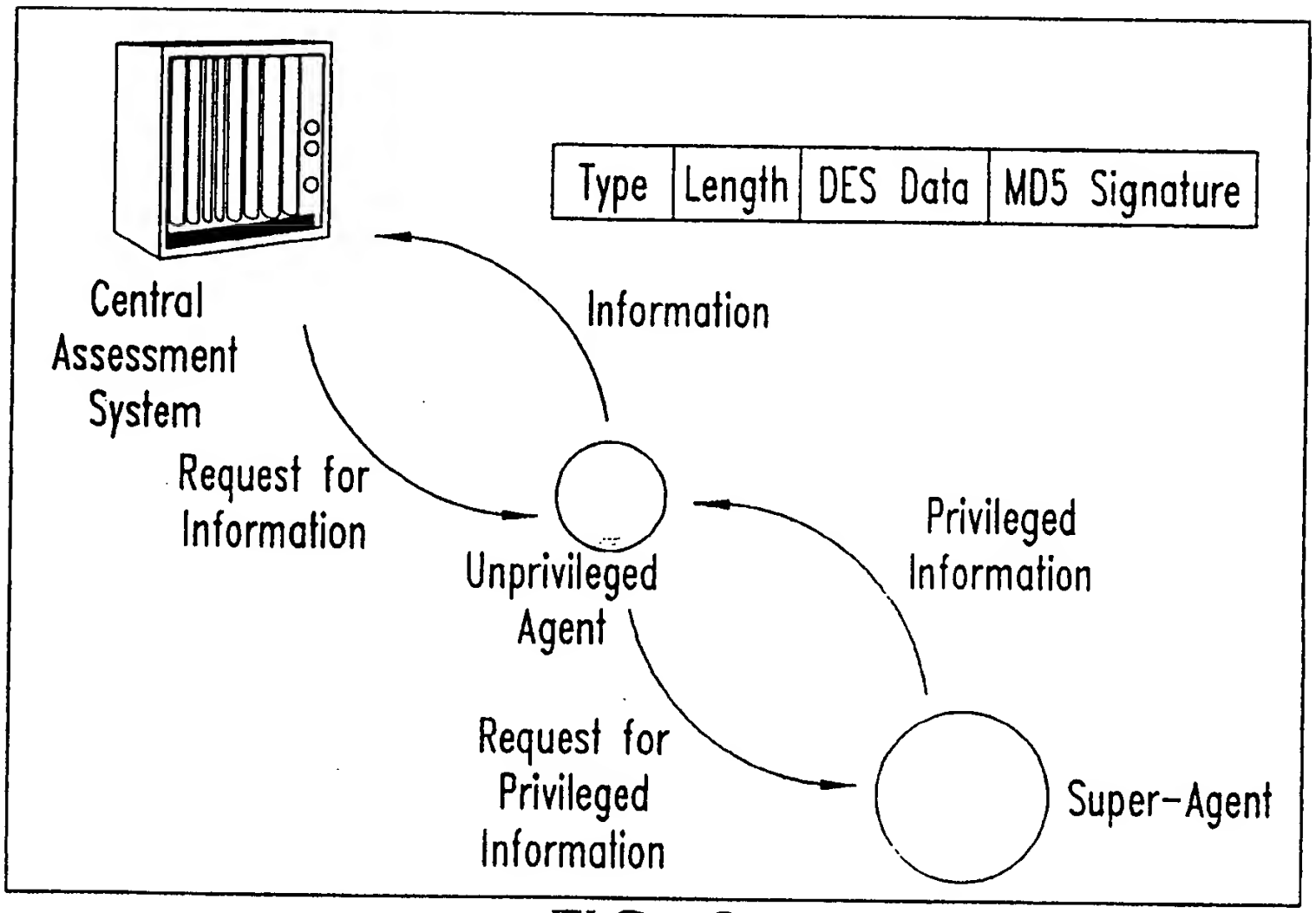

Type
Length
DES Data
MD5 Signature
Central
Assessment
System
Information
Request for
Information
Unprivileged
Agent
Privileged
Information
Request for
Privileged
Information
Super-Agent

FIG. 3

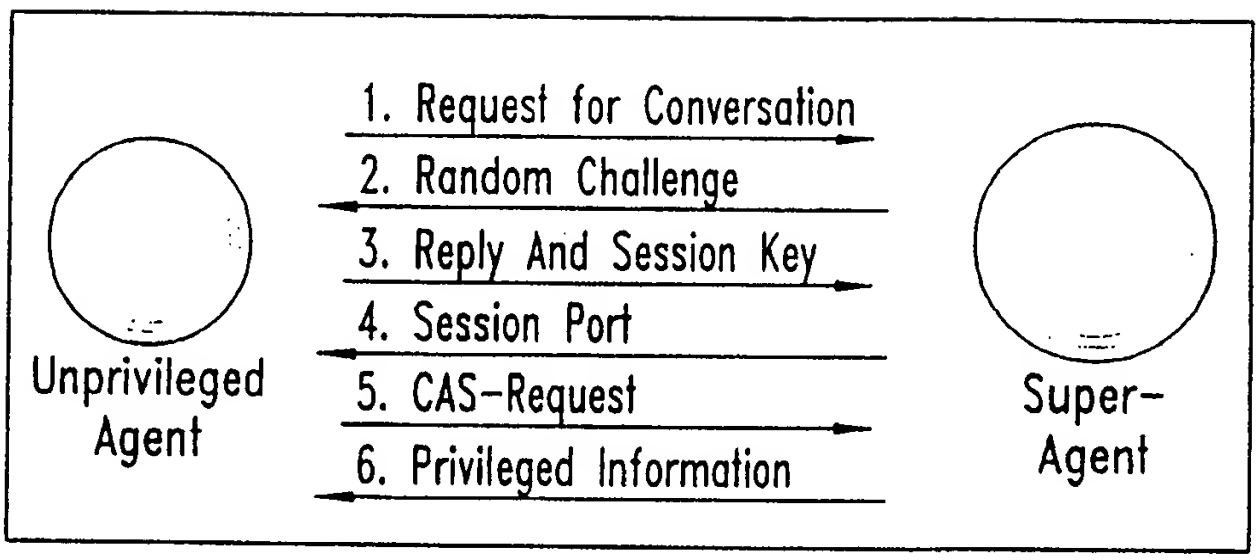

1. Request for Conversation
2. Random Challenge
3. Reply And Session Key
4. Session Port
5. CAS-Request
6. Privileged Information
Unprivileged Agent
Super-Agent

FIG. 4

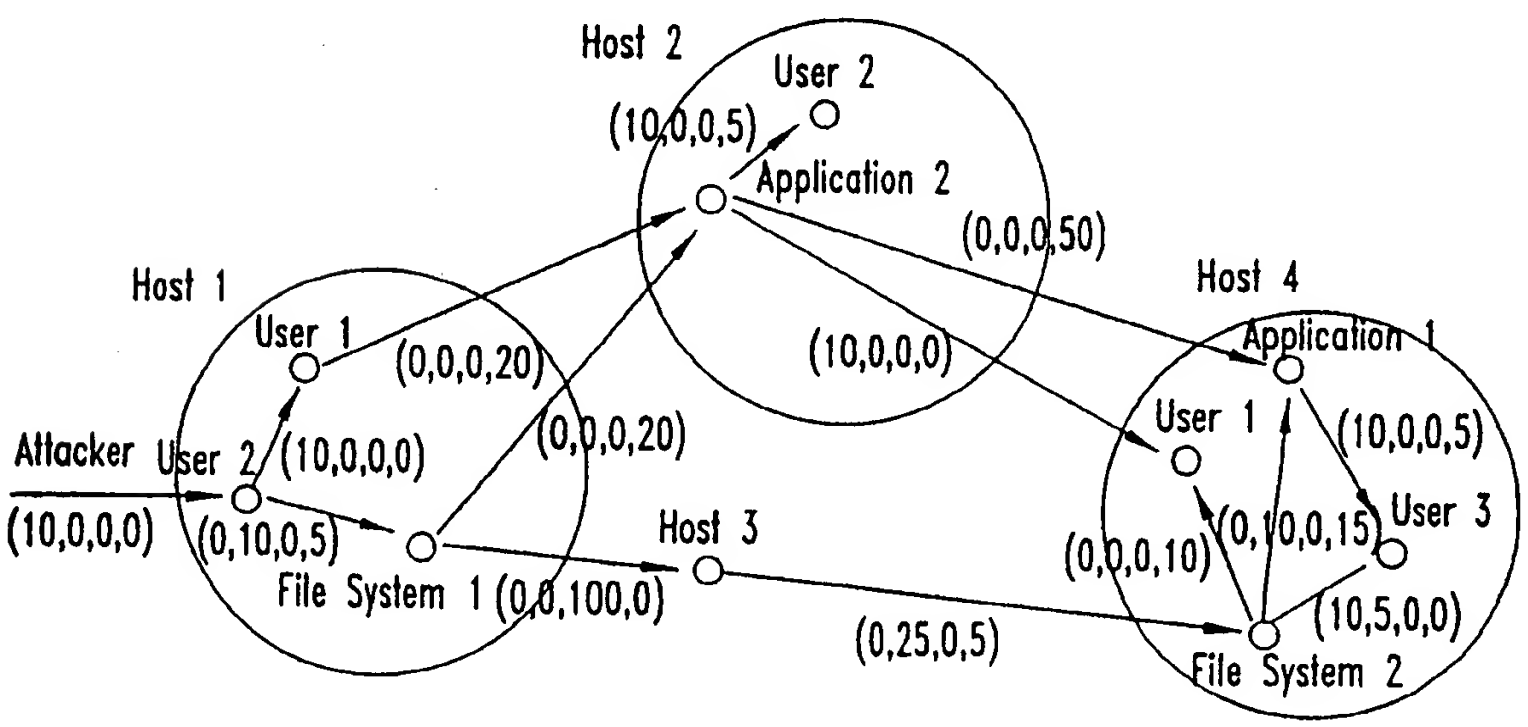

Host 2
User 2
(10,0,0,5)
Application 2
(0,0,0,50)
Host 1
User 1
(0,0,0,20)
(10,0,0,0)
Host 4
Application 1
Attacker
User 2
(10,0,0,0)
(0,0,0,20)
User 1
(10,0,0,5)
(10,0,0,0)
(0,10,0,5)
Host 3
(0,0,0,10)
(0,10,0,15)
User 3
File System 1 (0,0,100,0)
(0,25,0,5)
(10,5,0,0)
File System 2

FIG. 5

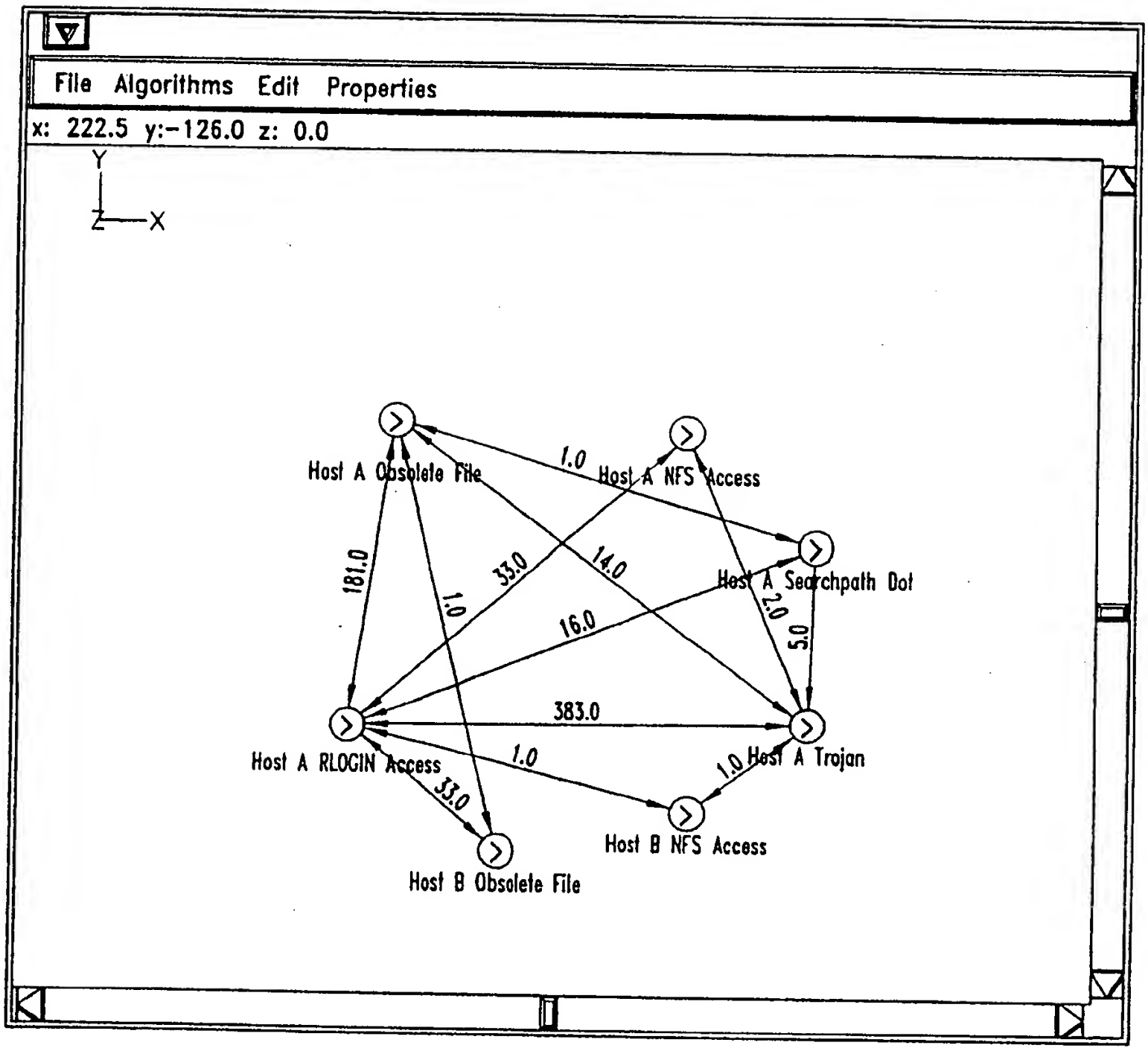

File Algorithms Edit Properties
x: 222.5 y:-126.0 z: 0.0
Y
Z
X
Host A Obsolete File
1.0
Host A NFS Access
181.0
33.0
14.0
1.0
16.0
2.0
5.0
Host A Searchpath Dot
383.0
Host A RLOGIN Access
1.0
1.0
Host A Trojan
33.0
Host B NFS Access
Host B Obsolete File

FIG. 6

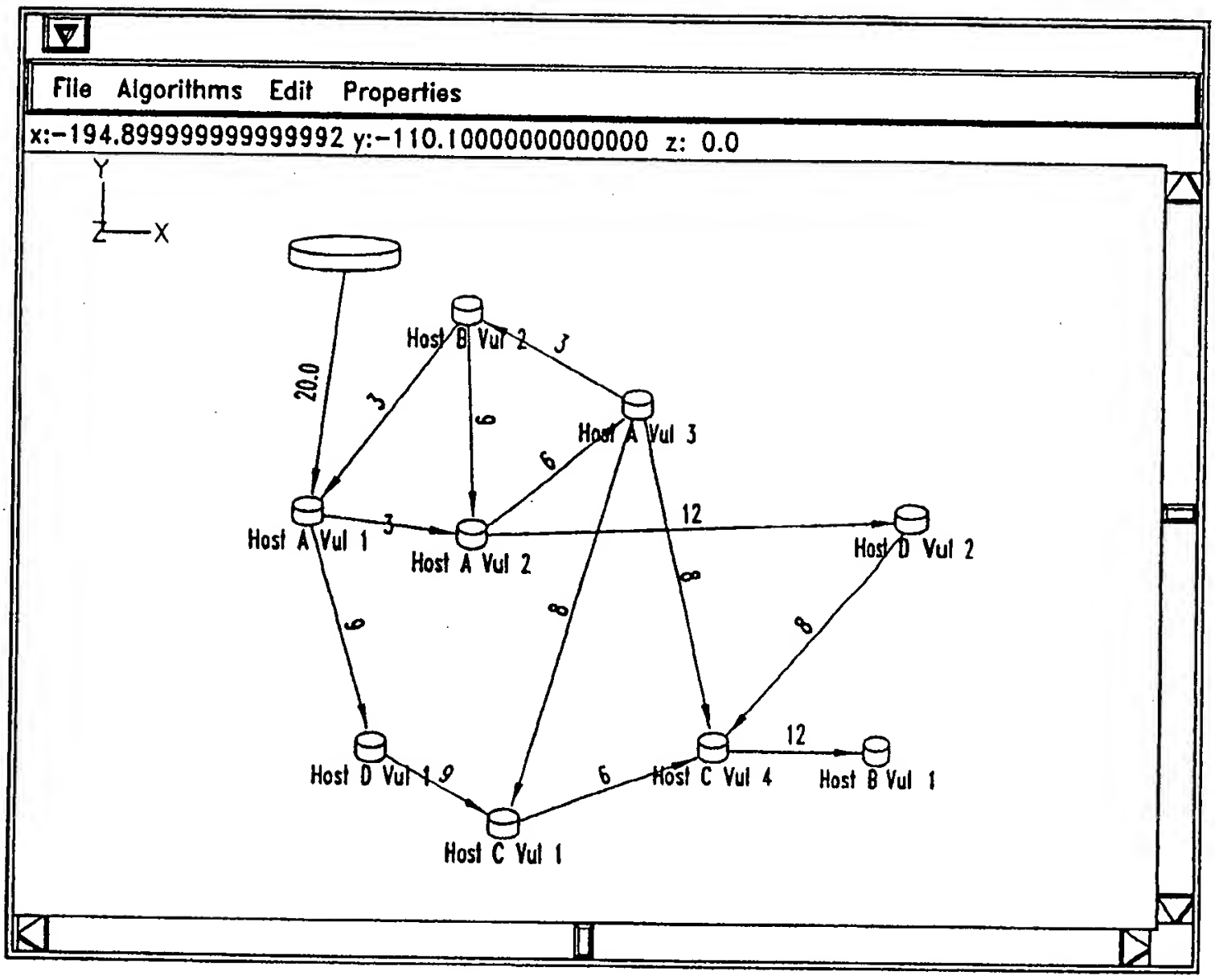

File Algorithms Edit Properties
x:-194.899999999999992 y:-110.10000000000000 z: 0.0
Y
Z
X
Host B Vul 2
Host A Vul 3
Host A Vul 1
Host A Vul 2
Host D Vul 2
Host D Vul 1
Host C Vul 1
Host C Vul 4
Host B Vul 1
20.0
3
3
6
6
12
6
8
8
8
9
6
12

FIG. 7

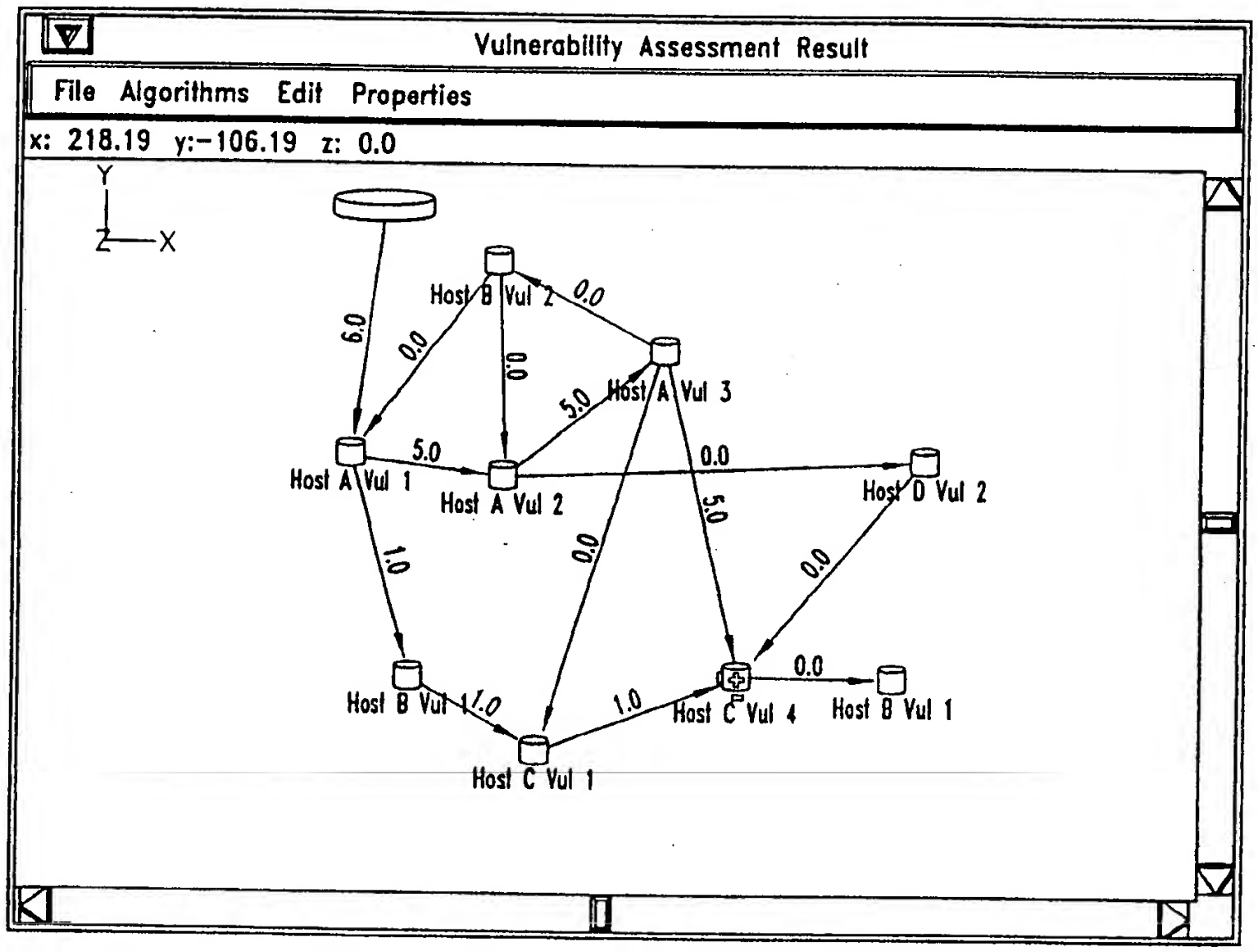

Vulnerability Assessment Result
File Algorithms Edit Properties
x: 218.19 y:-106.19 z: 0.0
Y
Z
X
Host B Vul 2
Host A Vul 3
Host A Vul 1
Host A Vul 2
Host D Vul 2
Host B Vul 1
Host C Vul 4
Host B Vul 1
Host C Vul 1
6.0
0.0
0.0
0.0
5.0
5.0
0.0
5.0
1.0
0.0
0.0
1.0
1.0
0.0

FIG. 8

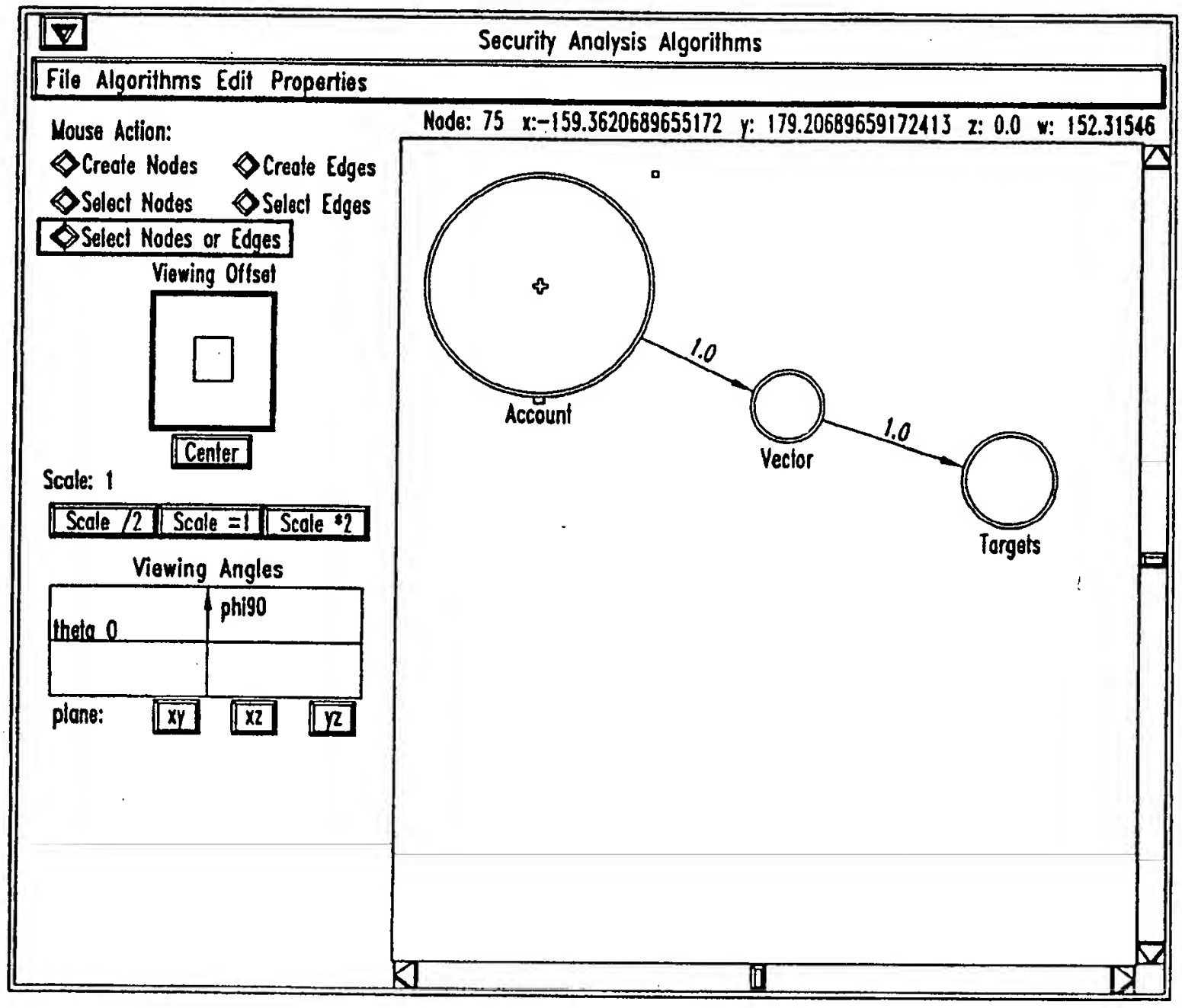

Security Analysis Algorithms
File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
theta 0
phi90
plane:
xy
xz
yz
Node: 75 x:-159.3620689655172 y: 179.20689659172413 z: 0.0 w: 152.31546
Account
1.0
Vector
1.0
Targets

FIG. 9

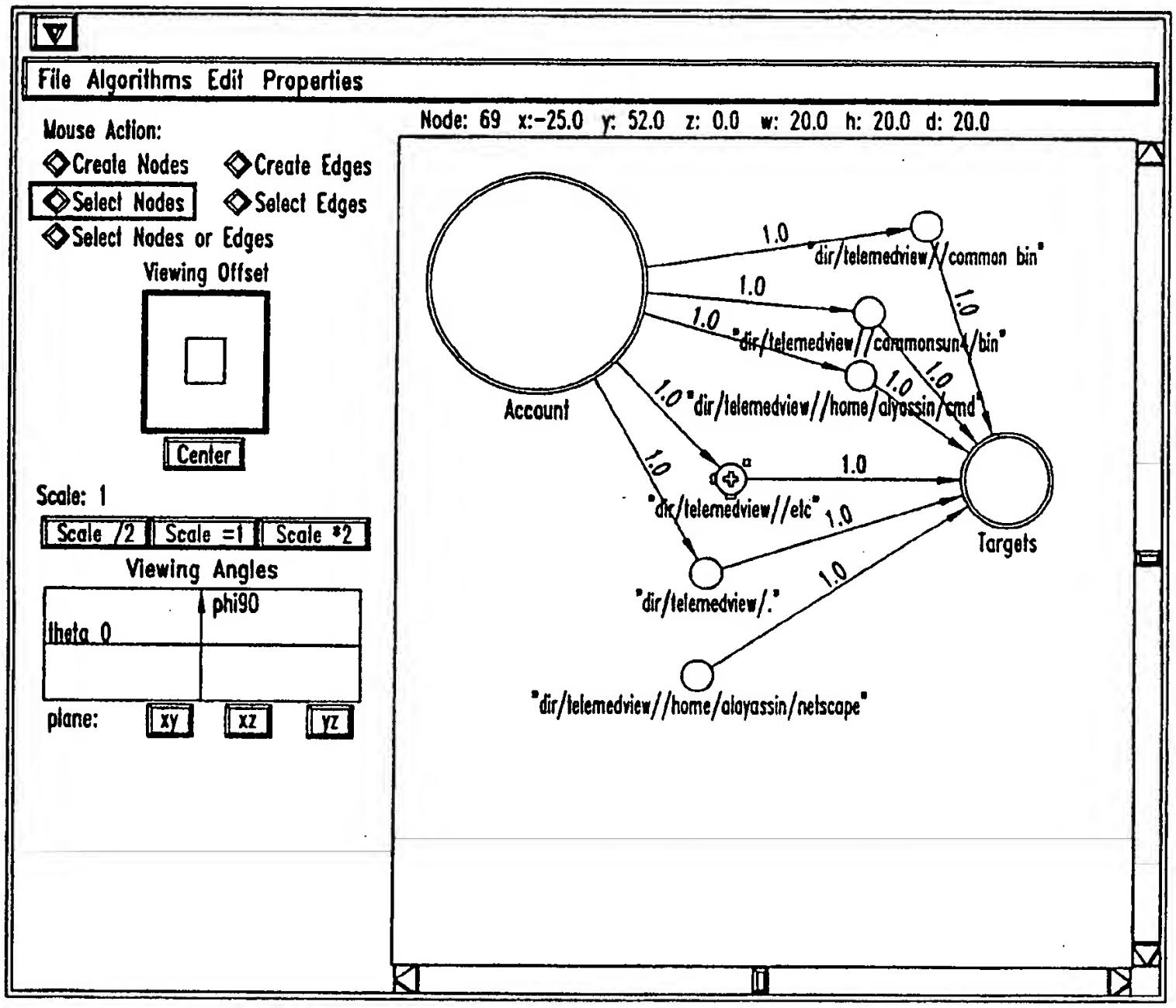

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
Node: 69 x:-25.0 y: 52.0 z: 0.0 w: 20.0 h: 20.0 d: 20.0
Account
1.0
"dir/telemedview//common bin"
"dir/telemedview//commonsun4/bin"
"dir/telemedview//home/alyassin/cmd"
"dir/telemedview//etc"
"dir/telemedview/."
"dir/telemedview//home/alyassin/netscape"
Targets

FIG. 10

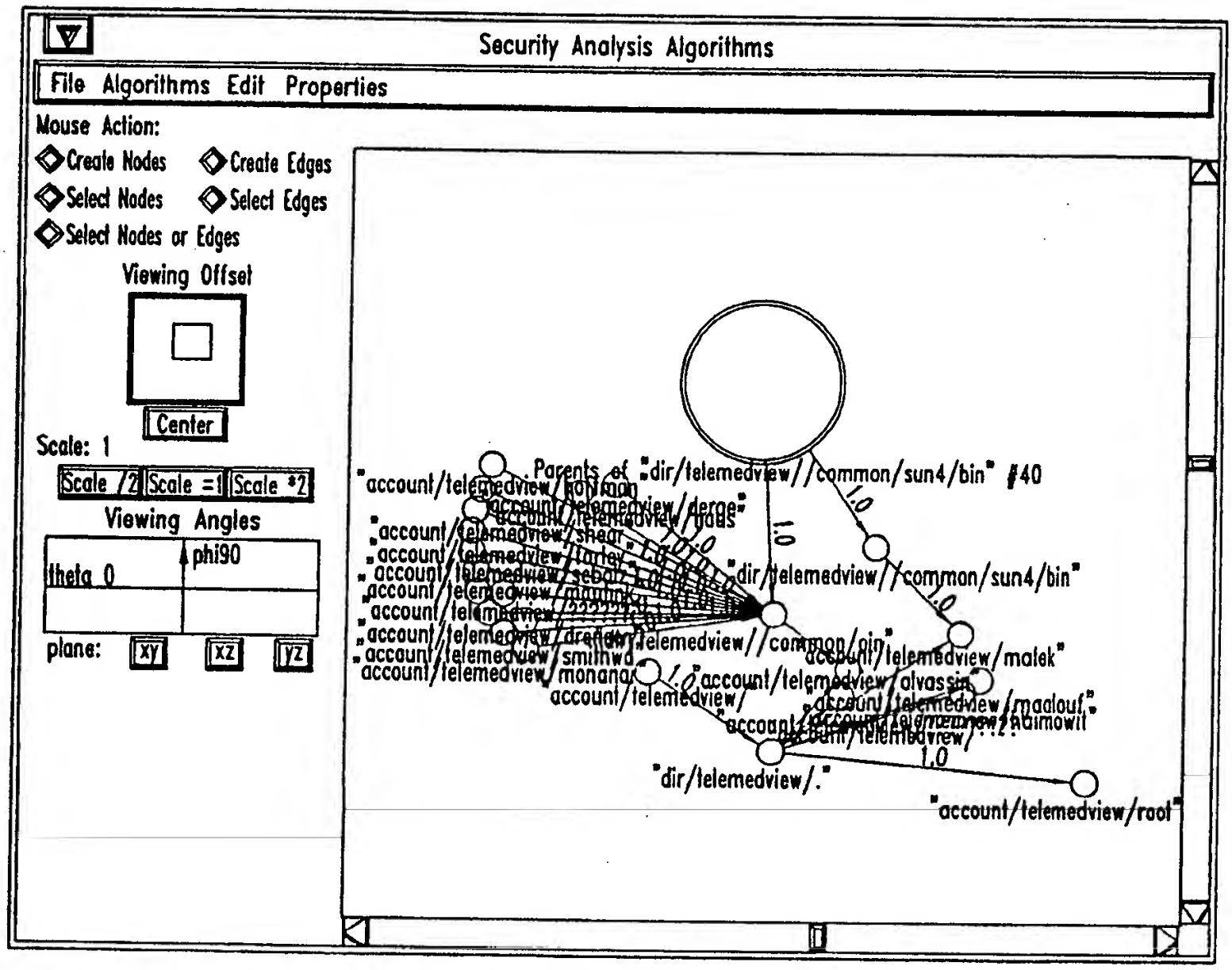

Security Analysis Algorithms
File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
theta 0
phi90
plane:
XY
XZ
YZ
Parents of "dir/telemedview//common/sun4/bin" #40
"account/telemedview/root"
"dir/telemedview/."
"dir/telemedview//common/sun4/bin"
"account/telemedview/malek"
"account/telemedview/alvassin"
"account/telemedview/maalouf"
1.0

FIG. 11

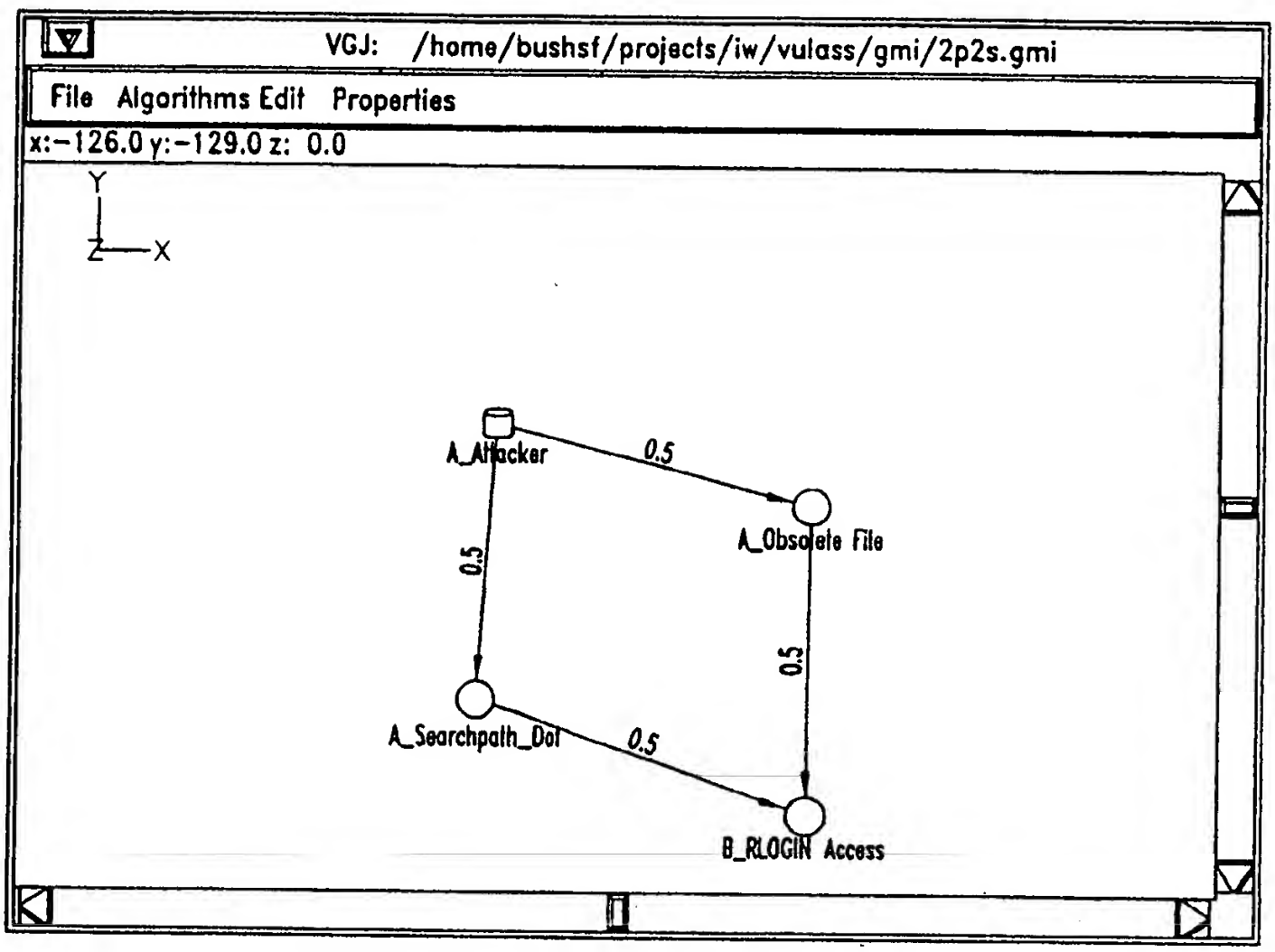

VGJ: /home/bushsf/projects/iw/vulass/gmi/2p2s.gmi
File Algorithms Edit Properties
x:-126.0 y:-129.0 z: 0.0
Y
Z
X
A_Attacker
0.5
A_Obsolete File
0.5
0.5
A_Searchpath_Dot
0.5
B_RLOGIN Access

FIG. 12

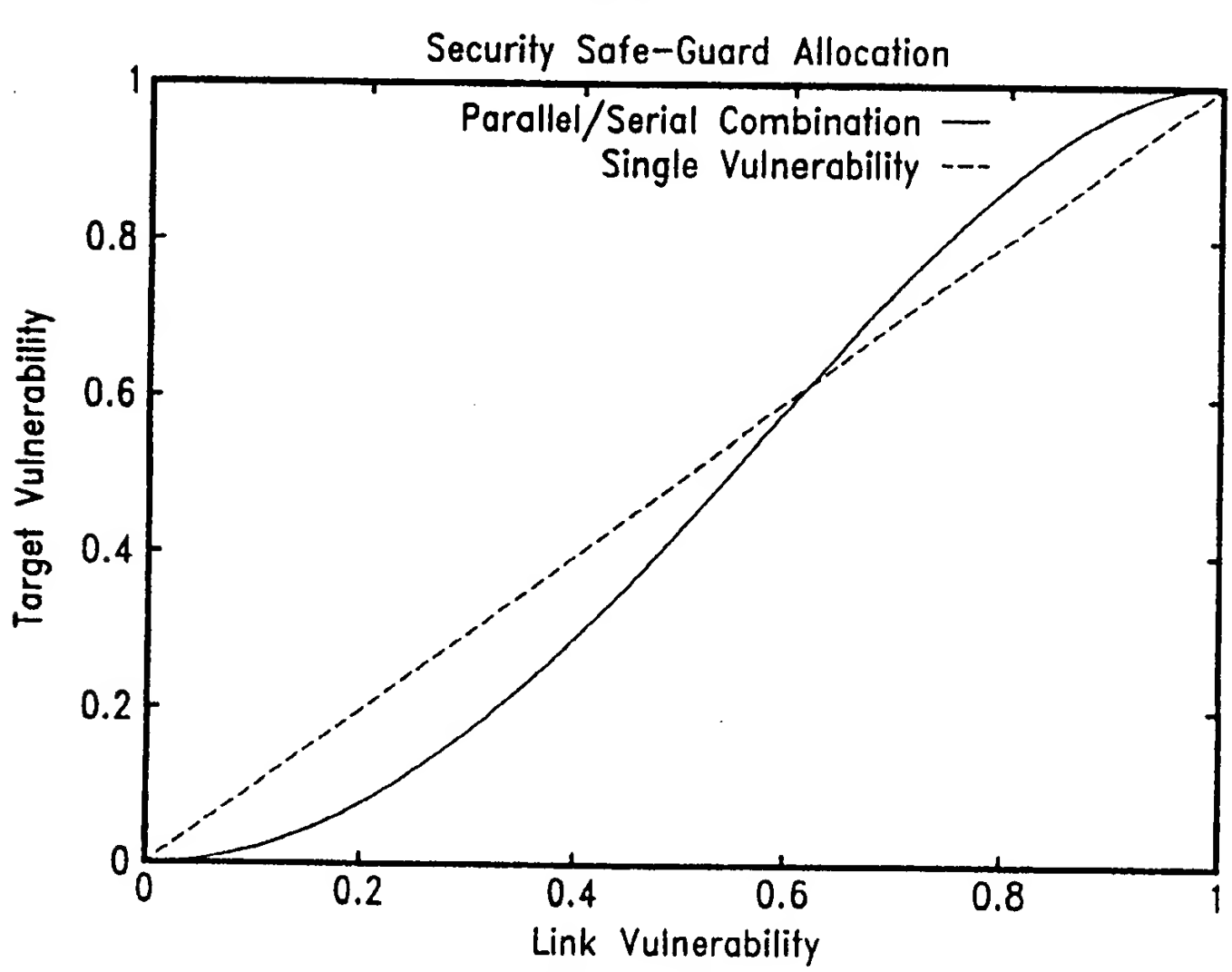

Security Safe-Guard Allocation
Parallel/Serial Combination
Single Vulnerability
Target Vulnerability
Link Vulnerability
0
0.2
0.4
0.6
0.8
1

FIG. 13

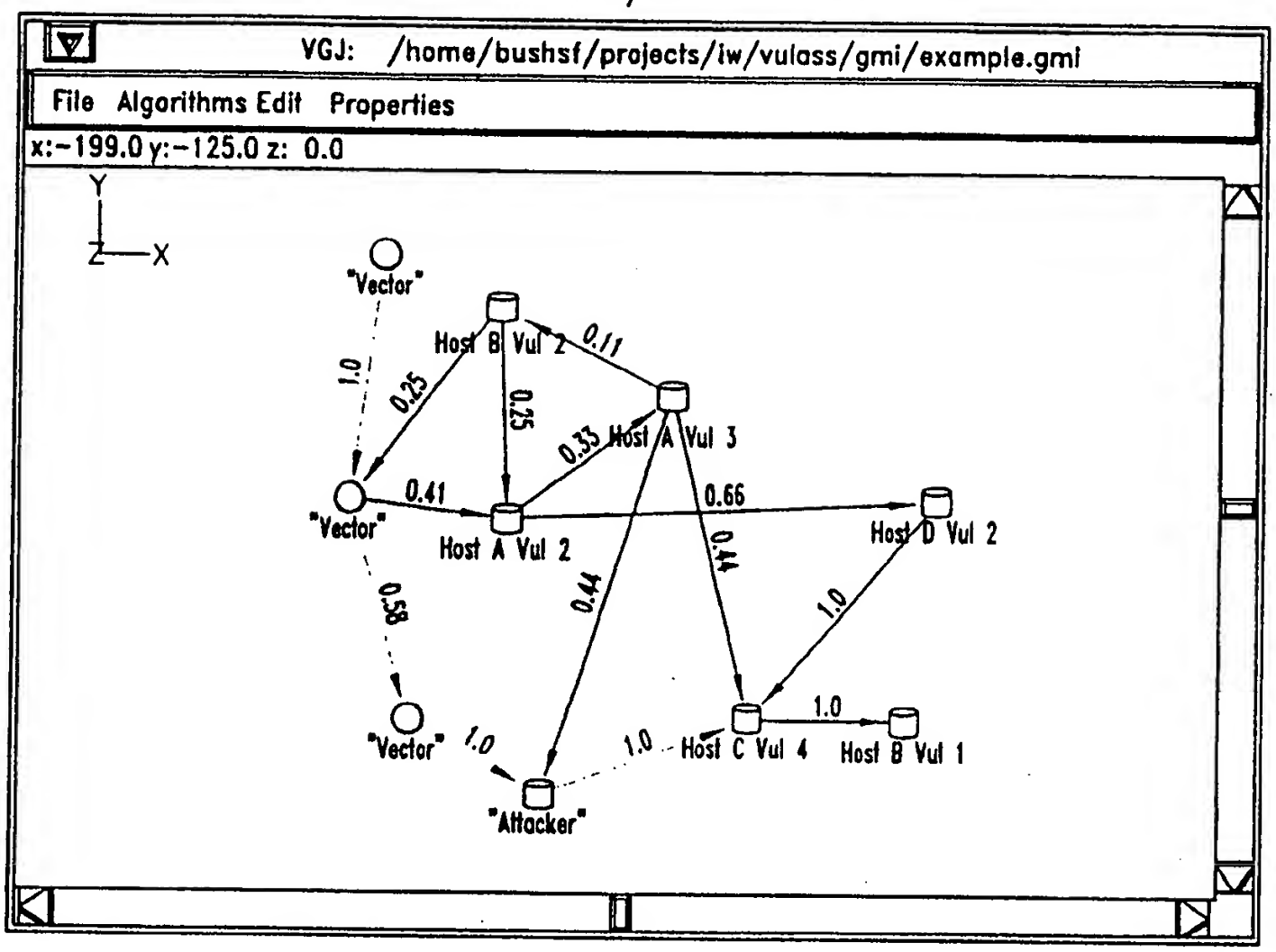

VGJ: /home/bushsf/projects/iw/vuloss/gmi/example.gmi
File Algorithms Edit Properties
x:-199.0 y:-125.0 z: 0.0
Y
Z
X
"Vector"
Host B Vul 2
Host A Vul 3
"Vector"
Host A Vul 2
Host D Vul 2
"Vector"
"Attacker"
Host C Vul 4
Host B Vul 1
1.0
0.25
0.25
0.11
0.33
0.41
0.66
0.58
0.44
0.44
1.0
1.0
1.0
1.0

FIG. 14

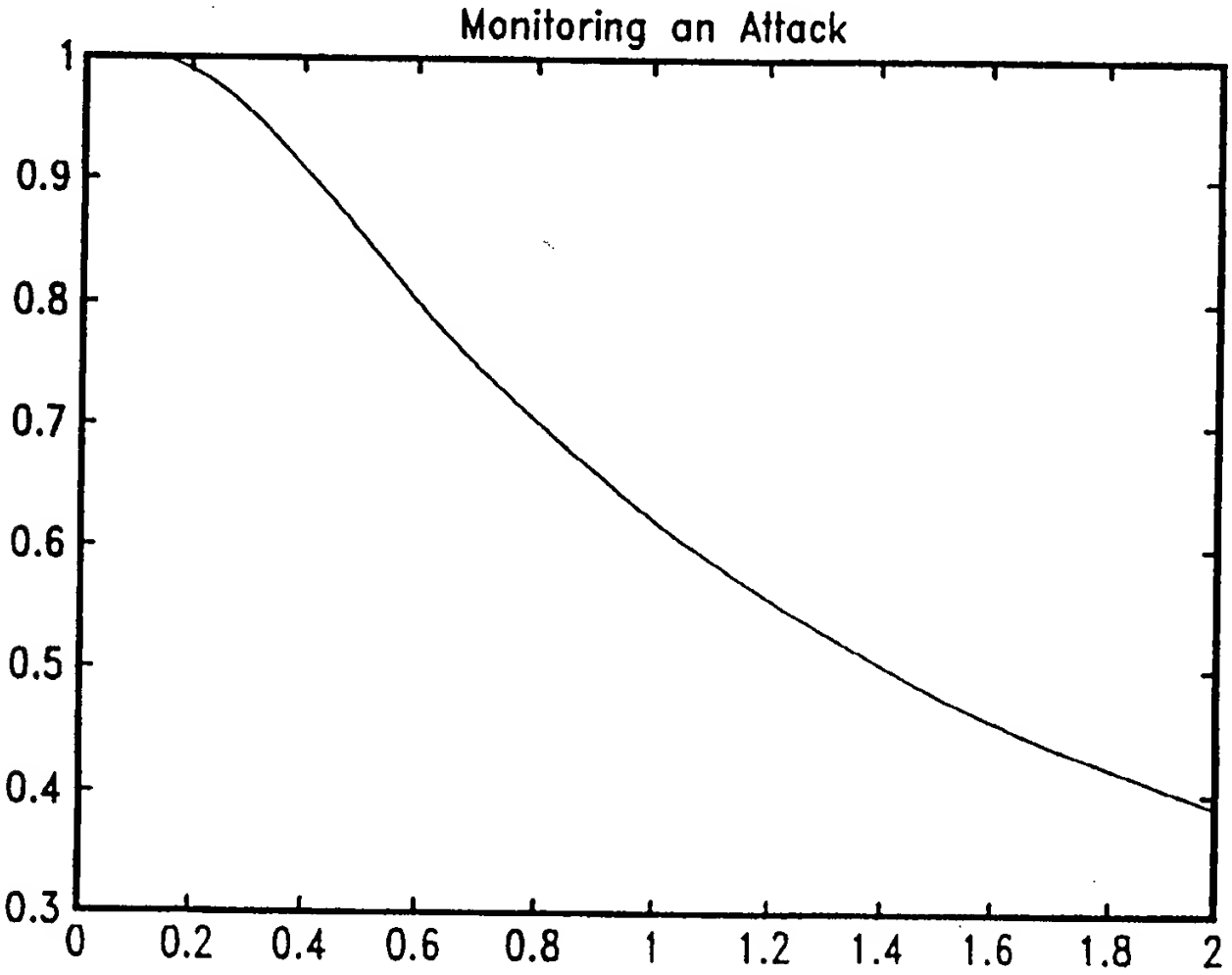

Monitoring an Attack
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

FIG. 15

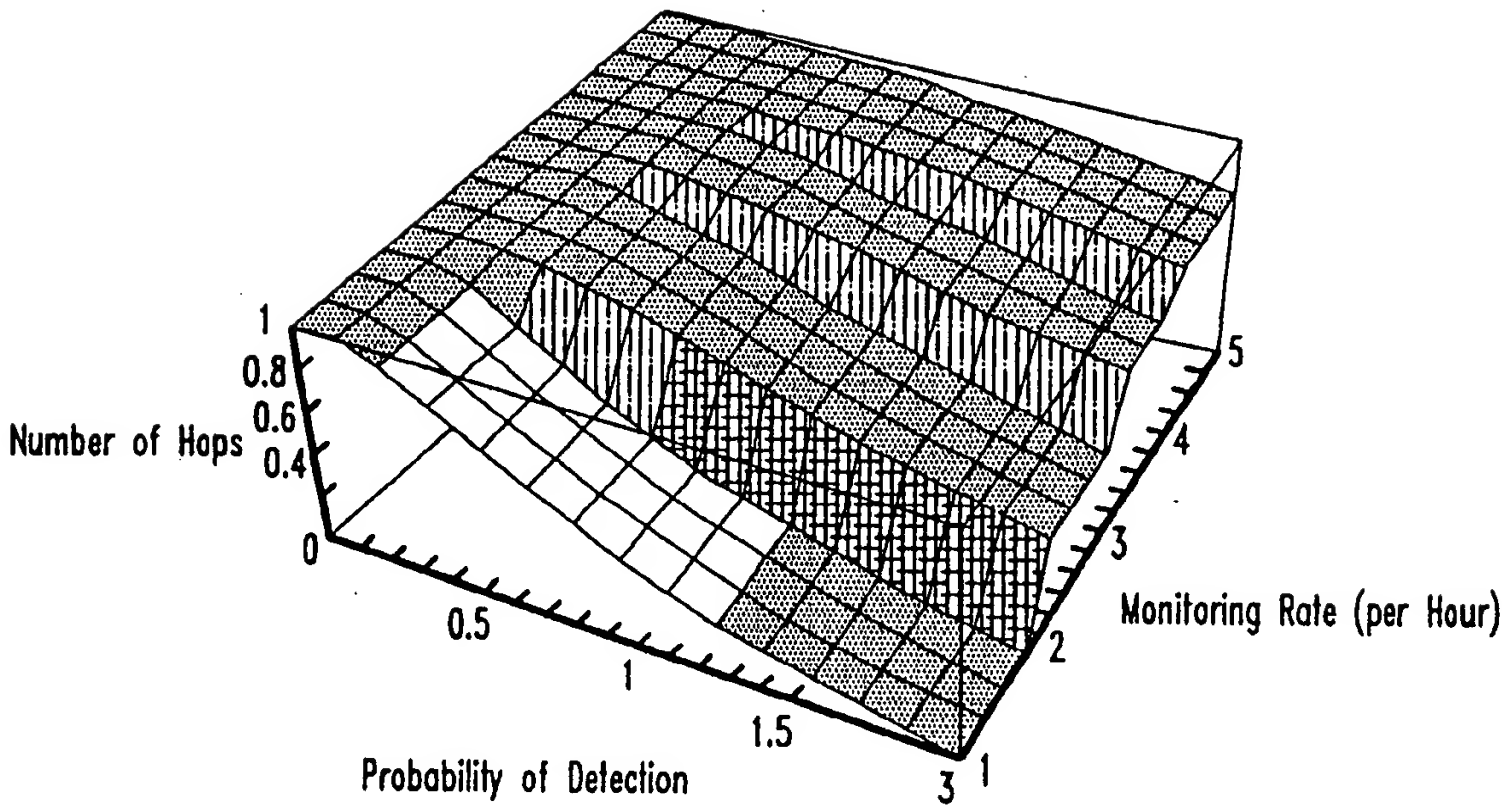

Number of Hops
1
0.8
0.6
0.4
0
0.5
1
1.5
Probability of Detection
1
2
3
4
5
Monitoring Rate (per Hour)
3
1

FIG. 16

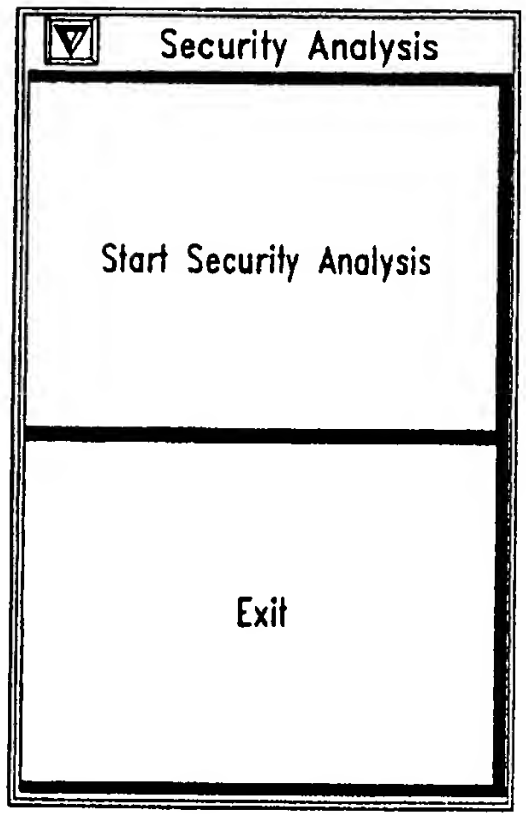

Security Analysis
Start Security Analysis
Exit

FIG. 17

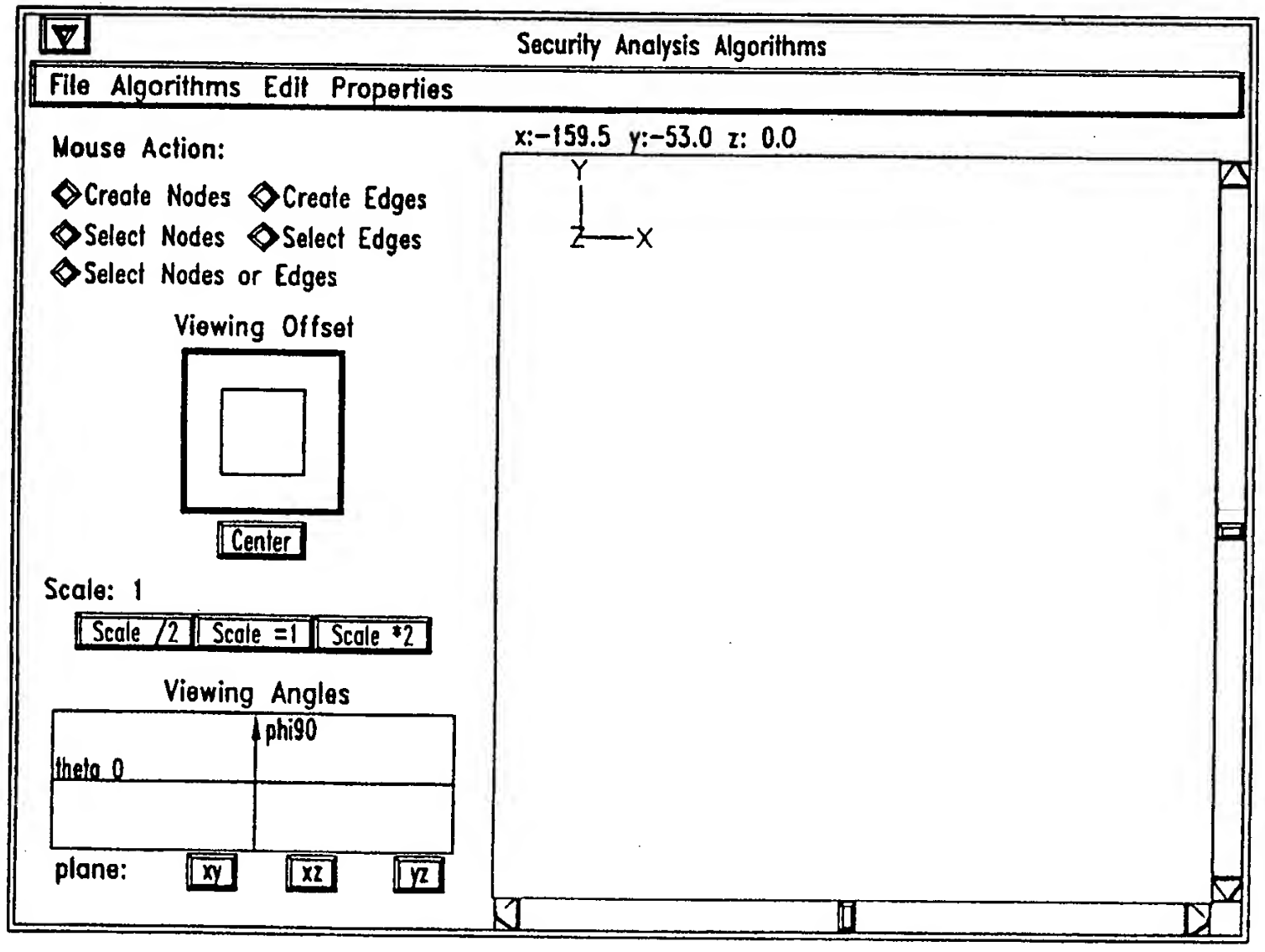

Security Analysis Algorithms
File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
x:-159.5 y:-53.0 z: 0.0
Y
Z
X

FIG. 18

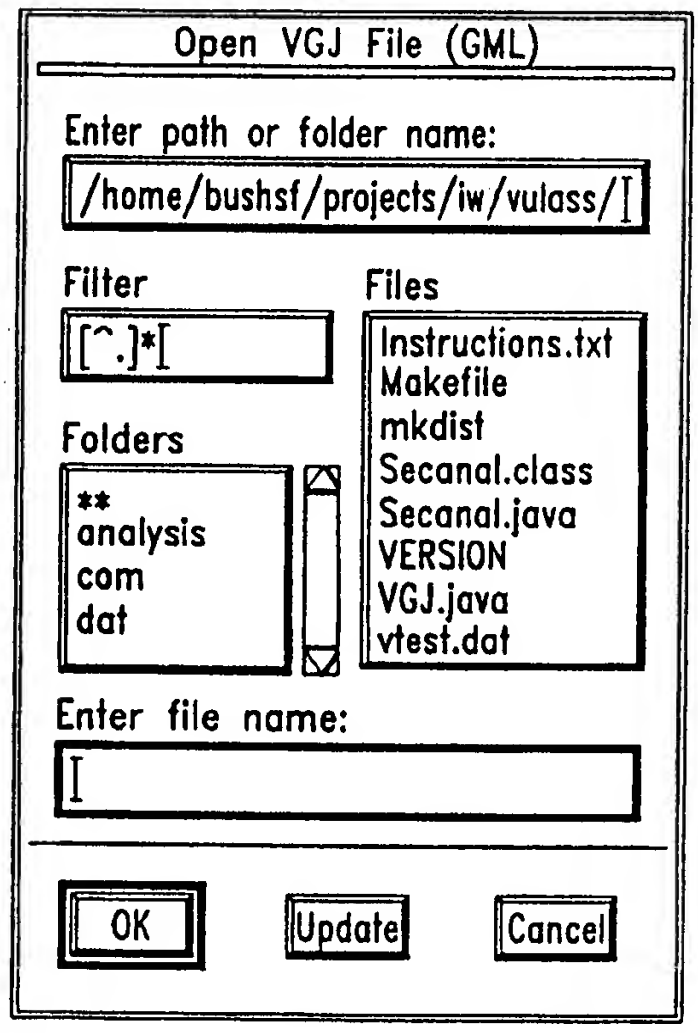

Open VGJ File (GML)
Enter path or folder name:
/home/bushsf/projects/iw/vulass/
Filter
[^.]*[
Files
Instructions.txt
Makefile
mkdist
Secanal.class
Secanal.java
VERSION
VGJ.java
vtest.dat
Folders
**
analysis
com
dat
Enter file name:
OK
Update
Cancel

FIG. 19

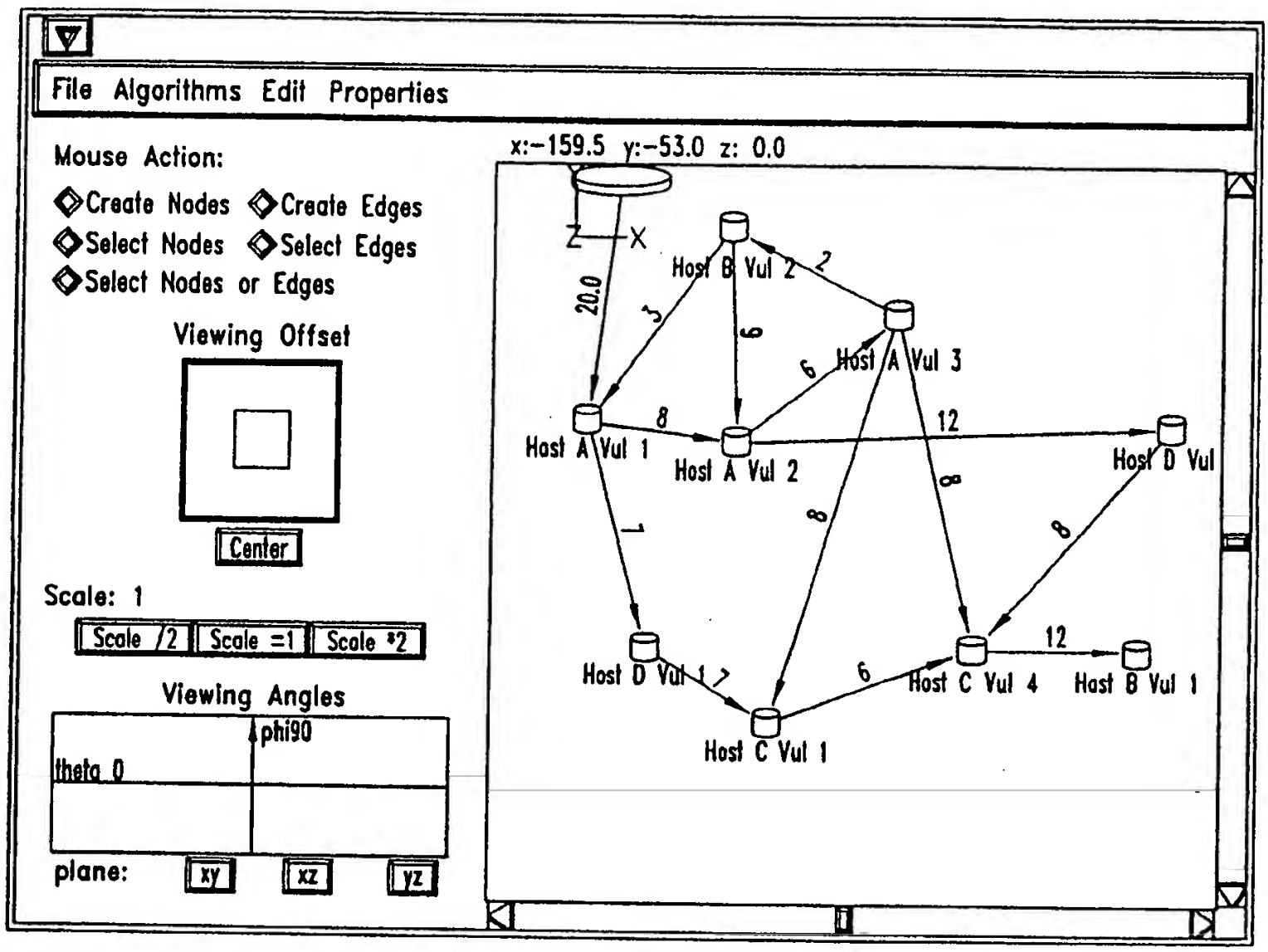

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
x:-159.5 y:-53.0 z: 0.0
Z
X
Host B Vul 2
Host A Vul 3
Host A Vul 1
Host A Vul 2
Host D Vul
Host D Vul 1
Host C Vul 4
Host B Vul 1
Host C Vul 1
20.0
3
2
6
6
8
12
1
8
8
8
12
6

FIG. 20

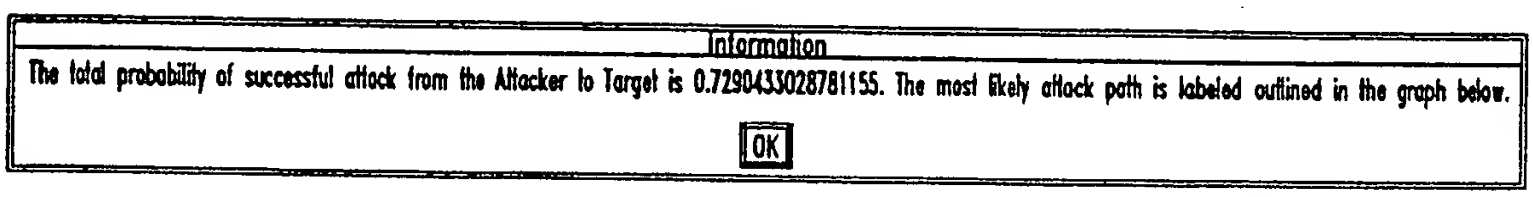

Information
The total probability of successful attack from the Attacker to Target is 0.7290433028781155. The most likely attack path is labeled outlined in the graph below.
OK

FIG. 21

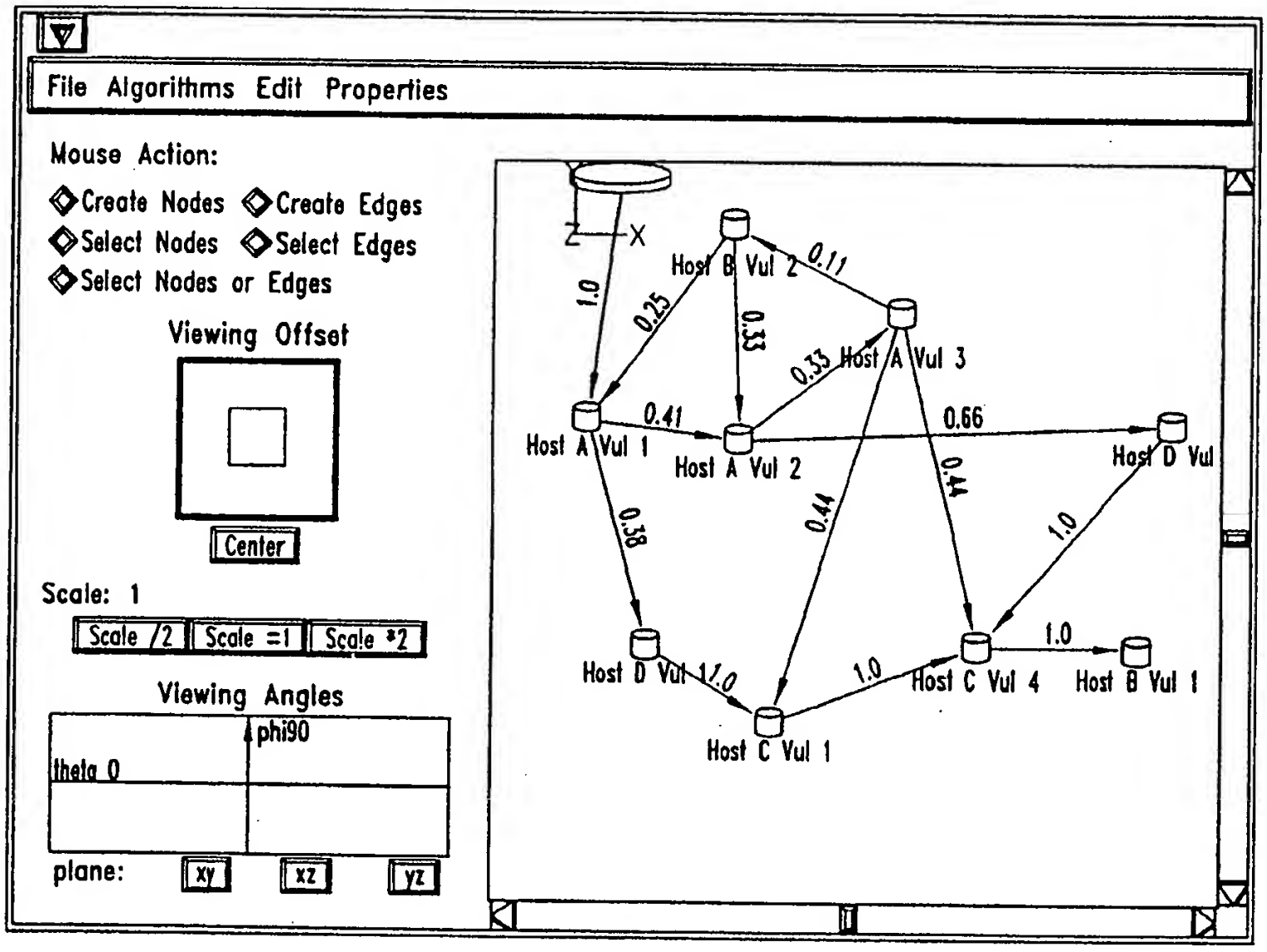

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
XY
XZ
YZ
Z
X
Host B Vul 2
Host A Vul 3
Host A Vul 1
Host A Vul 2
Host D Vul
Host D Vul 1
Host C Vul 4
Host B Vul 1
Host C Vul 1
1.0
0.25
0.33
0.11
0.33
0.41
0.66
0.38
0.44
0.44
1.0
1.0
1.0
1.0

FIG. 22

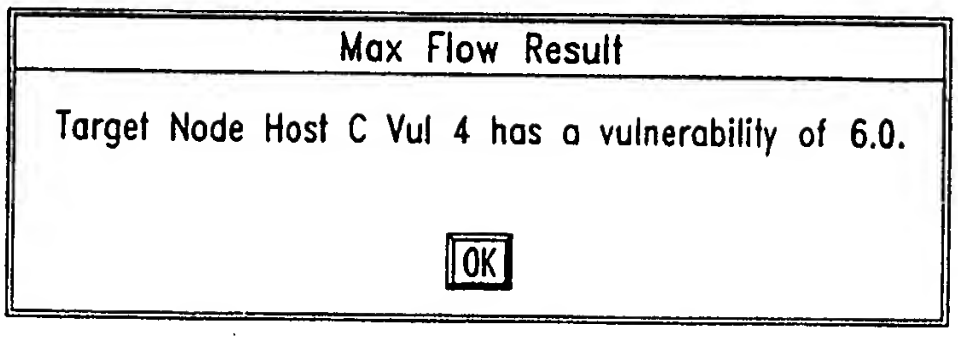

Max Flow Result
Target Node Host C Vul 4 has a vulnerability of 6.0.
OK

FIG. 23

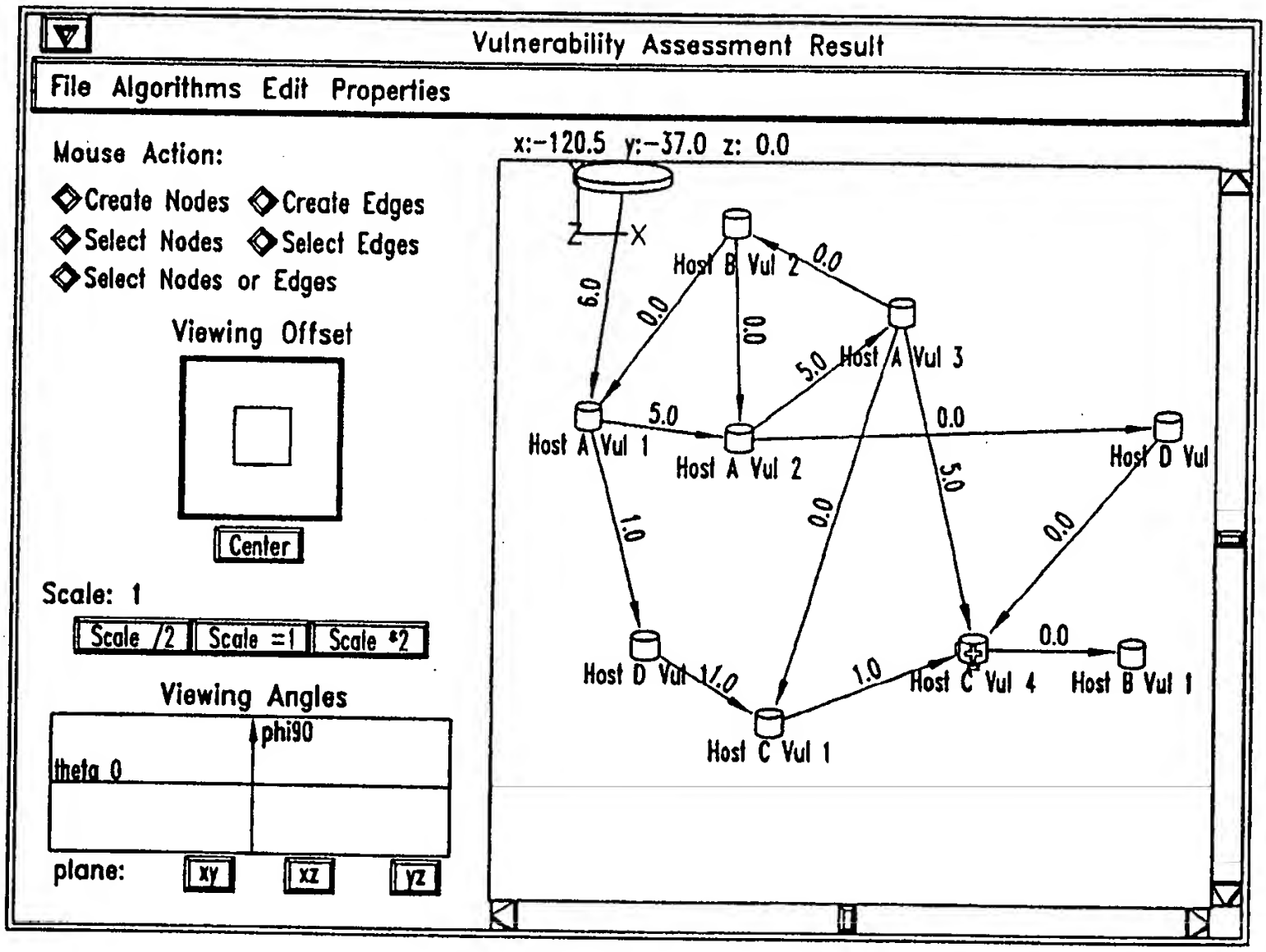

Vulnerability Assessment Result
File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
x:-120.5 y:-37.0 z: 0.0
Z
X
Host B Vul 2
Host A Vul 3
Host A Vul 1
Host A Vul 2
Host D Vul
Host D Vul
Host C Vul 4
Host B Vul 1
Host C Vul 1
6.0
0.0
0.0
0.0
5.0
5.0
0.0
1.0
0.0
5.0
0.0
1.0
1.0
0.0

FIG. 24

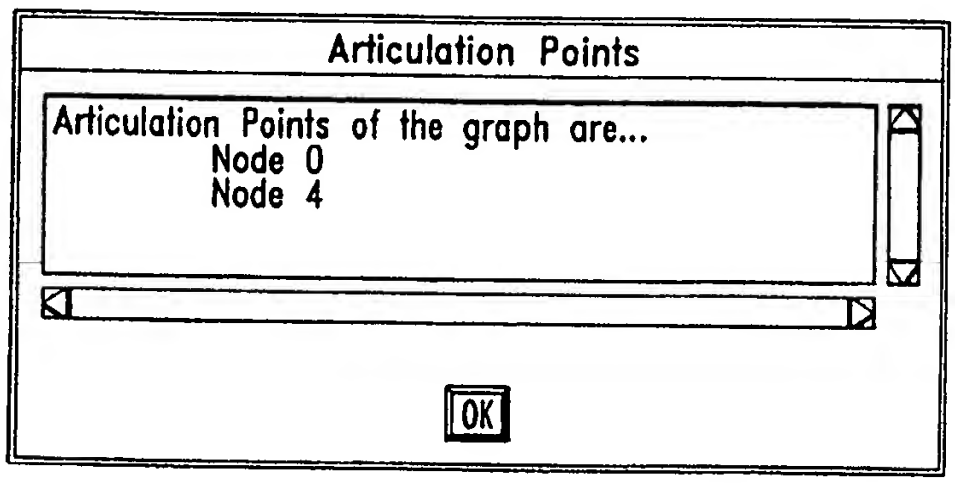

Articulation Points
Articulation Points of the graph are...
Node 0
Node 4
OK

FIG. 25

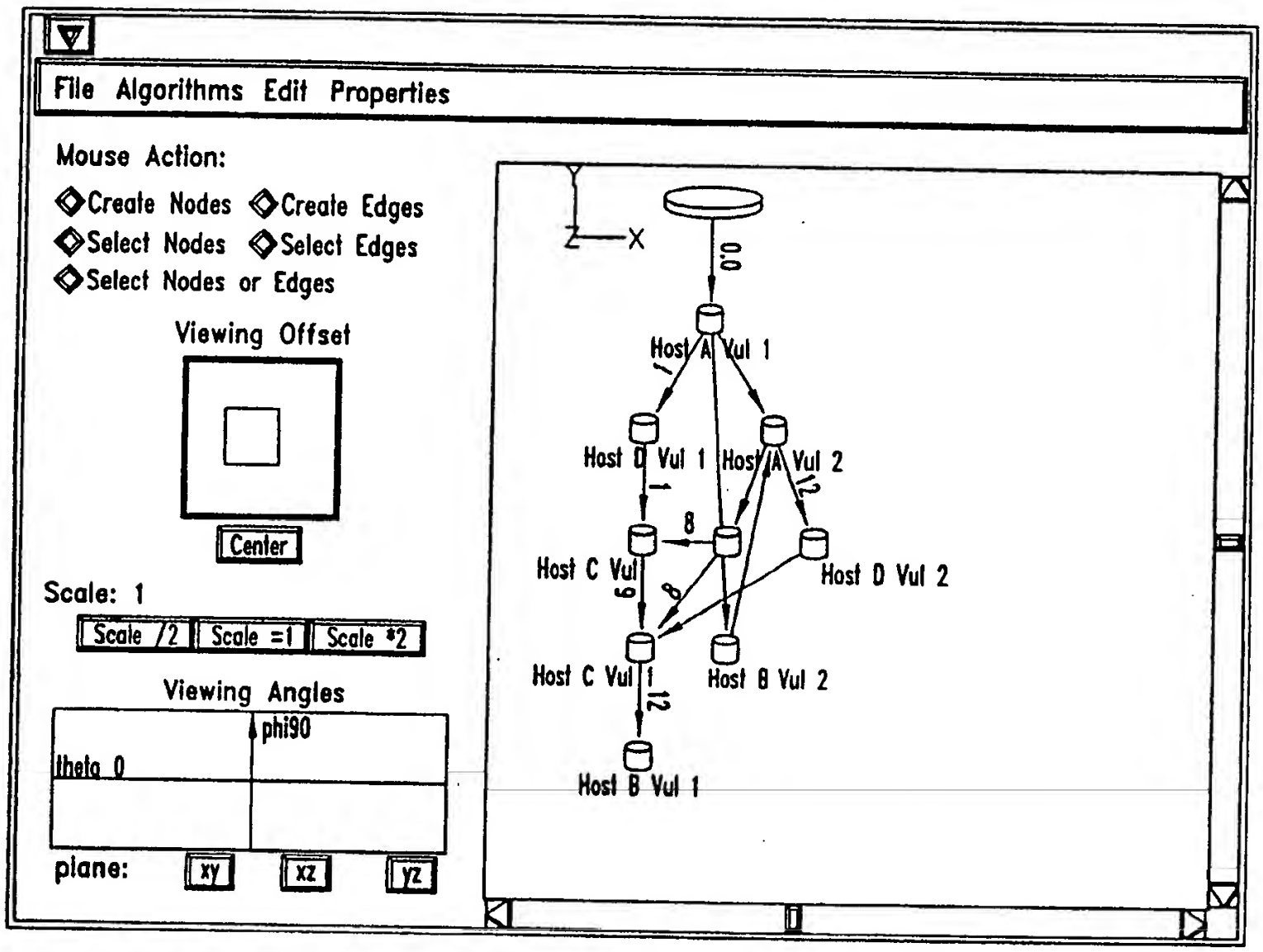

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
Z
X
Host A Vul 1
Host D Vul 1
Host A Vul 2
Host C Vul
Host D Vul 2
Host C Vul 1
Host B Vul 2
Host B Vul 1

FIG. 26

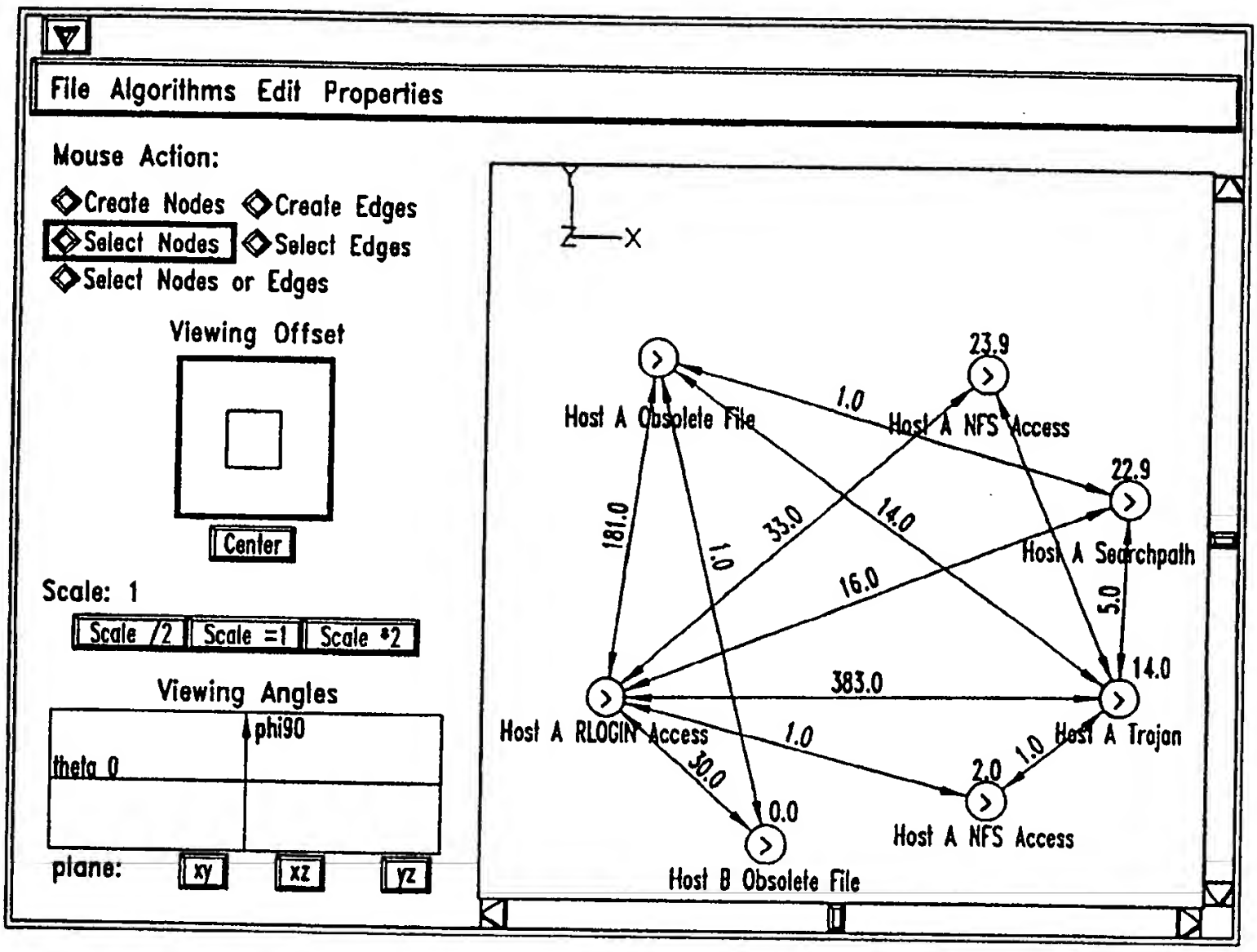

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
Z
X
Host A Obsolete File
23.9
1.0
Host A NFS Access
22.9
181.0
0.1
33.0
14.0
Host A Searchpath
16.0
5.0
14.0
383.0
Host A RLOGIN Access
1.0
Host A Trojan
30.0
2.0
1.0
0.0
Host A NFS Access
Host B Obsolete File

FIG. 27

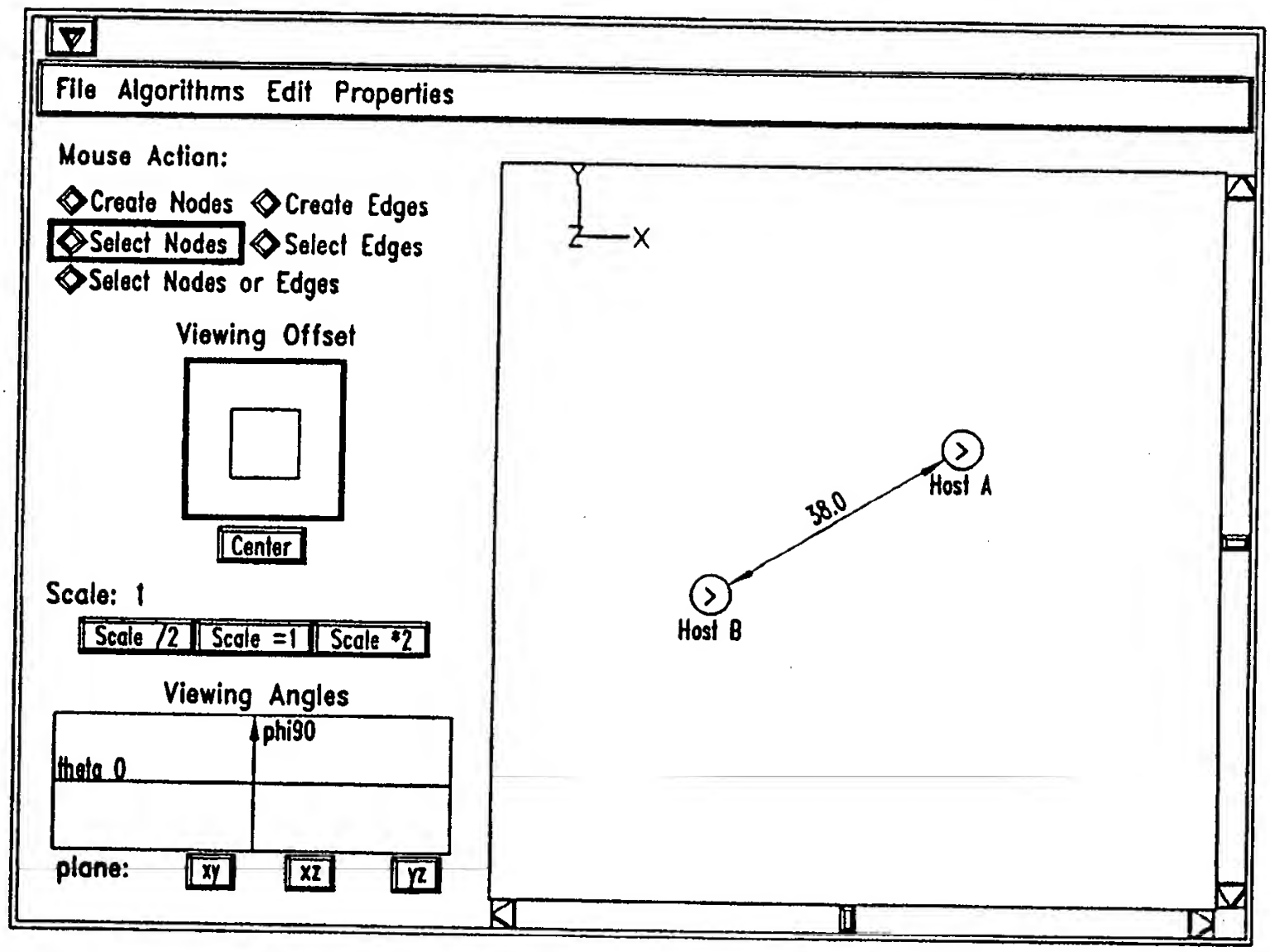

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
Z
X
38.0
Host A
Host B

FIG. 28

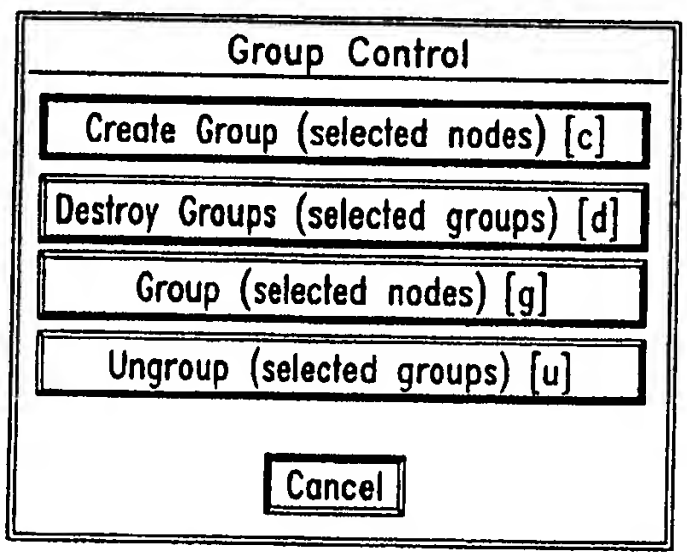

Group Control
Create Group (selected nodes) [c]
Destroy Groups (selected groups) [d]
Group (selected nodes) [g]
Ungroup (selected groups) [u]
Cancel

FIG. 29

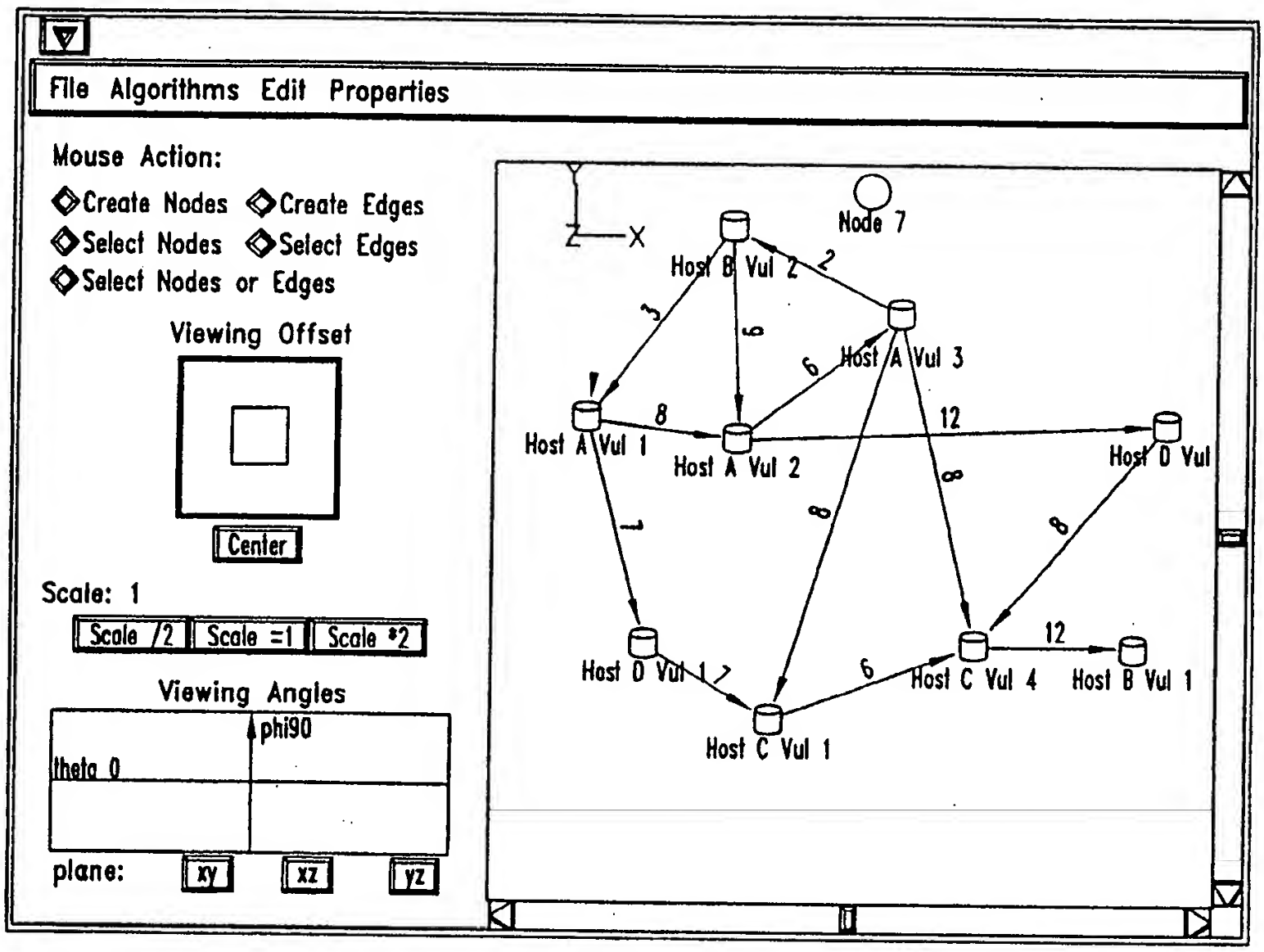

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
Z
X
Node 7
Host B Vul 2
Host A Vul 3
Host A Vul 1
Host A Vul 2
Host D Vul
Host D Vul 1
Host C Vul 4
Host B Vul 1
Host C Vul 1
2
3
5
6
8
12
1
8
8
8
7
6
12

FIG. 30

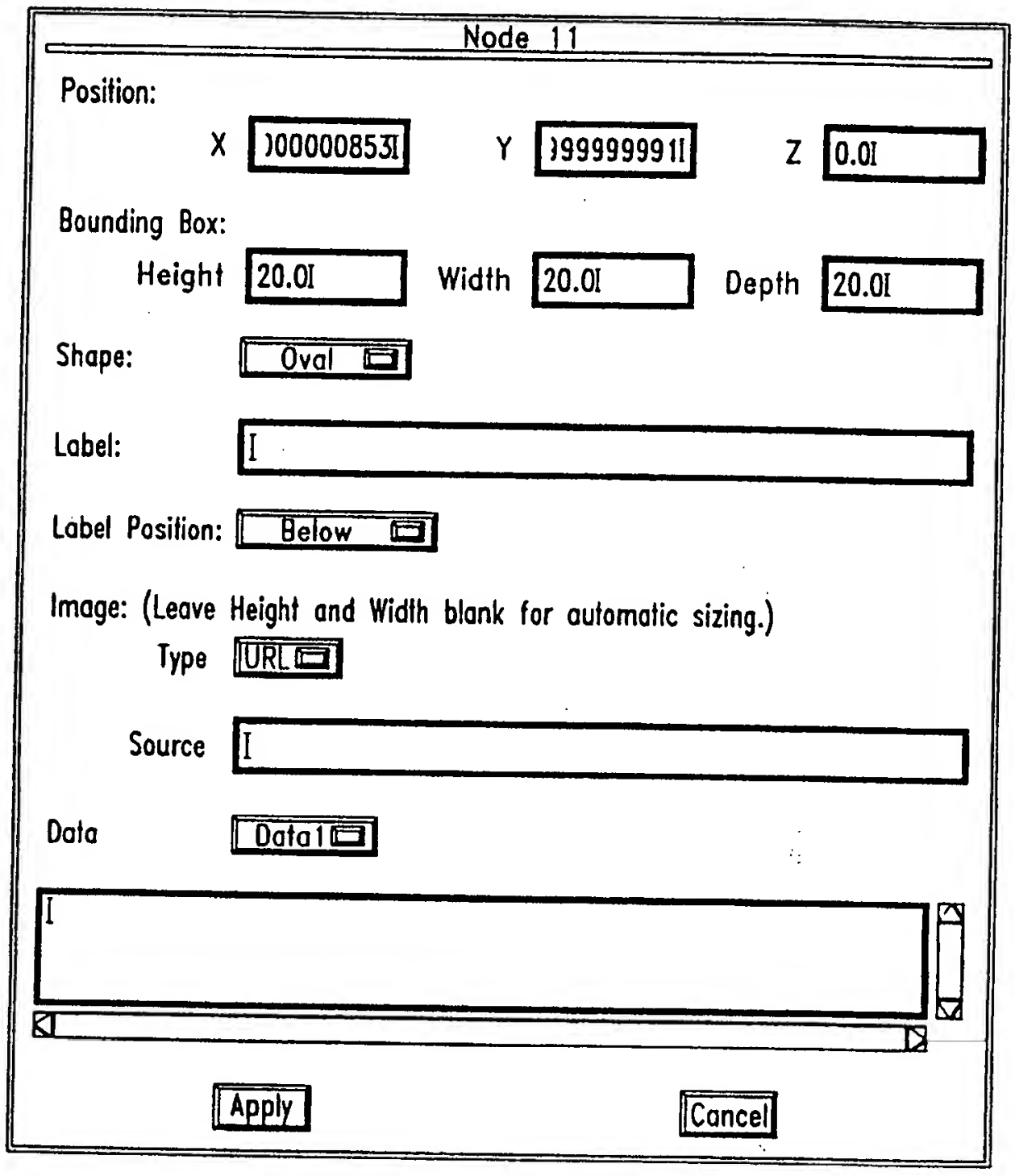

Node 11
Position:
X
)00000853I
Y
)99999991I
Z
0.0I
Bounding Box:
Height
20.0I
Width
20.0I
Depth
20.0I
Shape:
Oval
Label:
I
Label Position:
Below
Image: (Leave Height and Width blank for automatic sizing.)
Type
URL
Source
I
Data
Data1
I
Apply
Cancel

FIG. 31

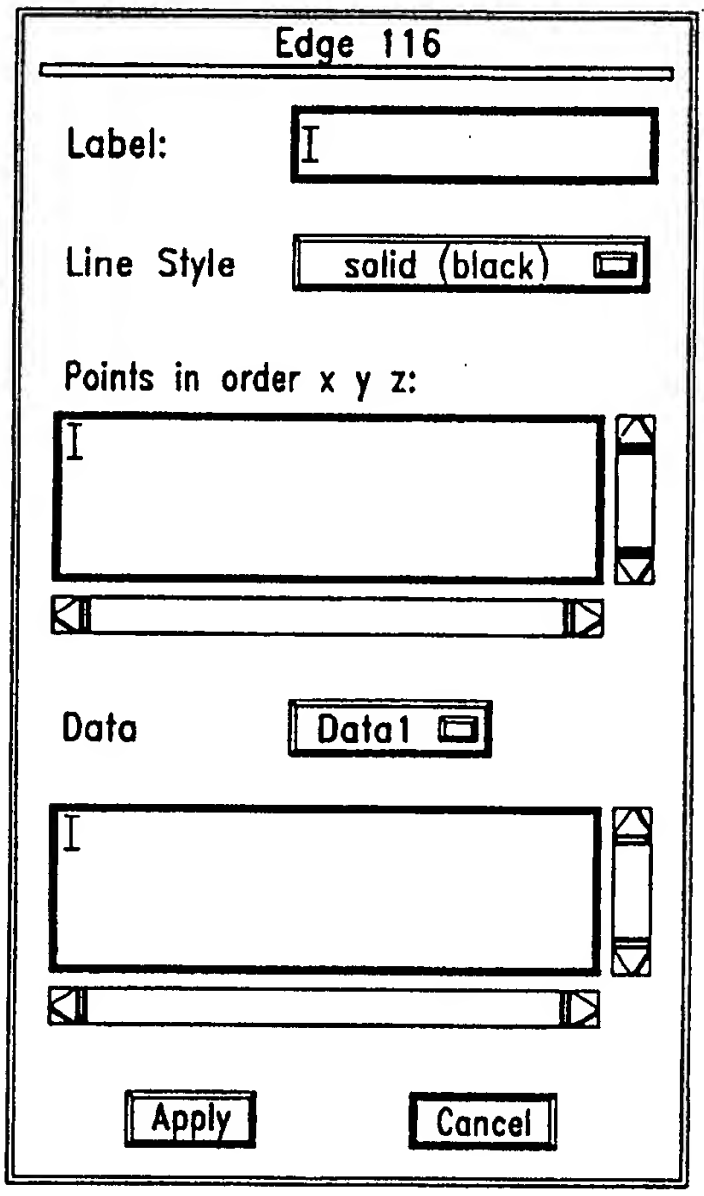

Edge 116
Label:
Line Style
solid (black)
Points in order x y z:
Data
Data1
Apply
Cancel

FIG. 32

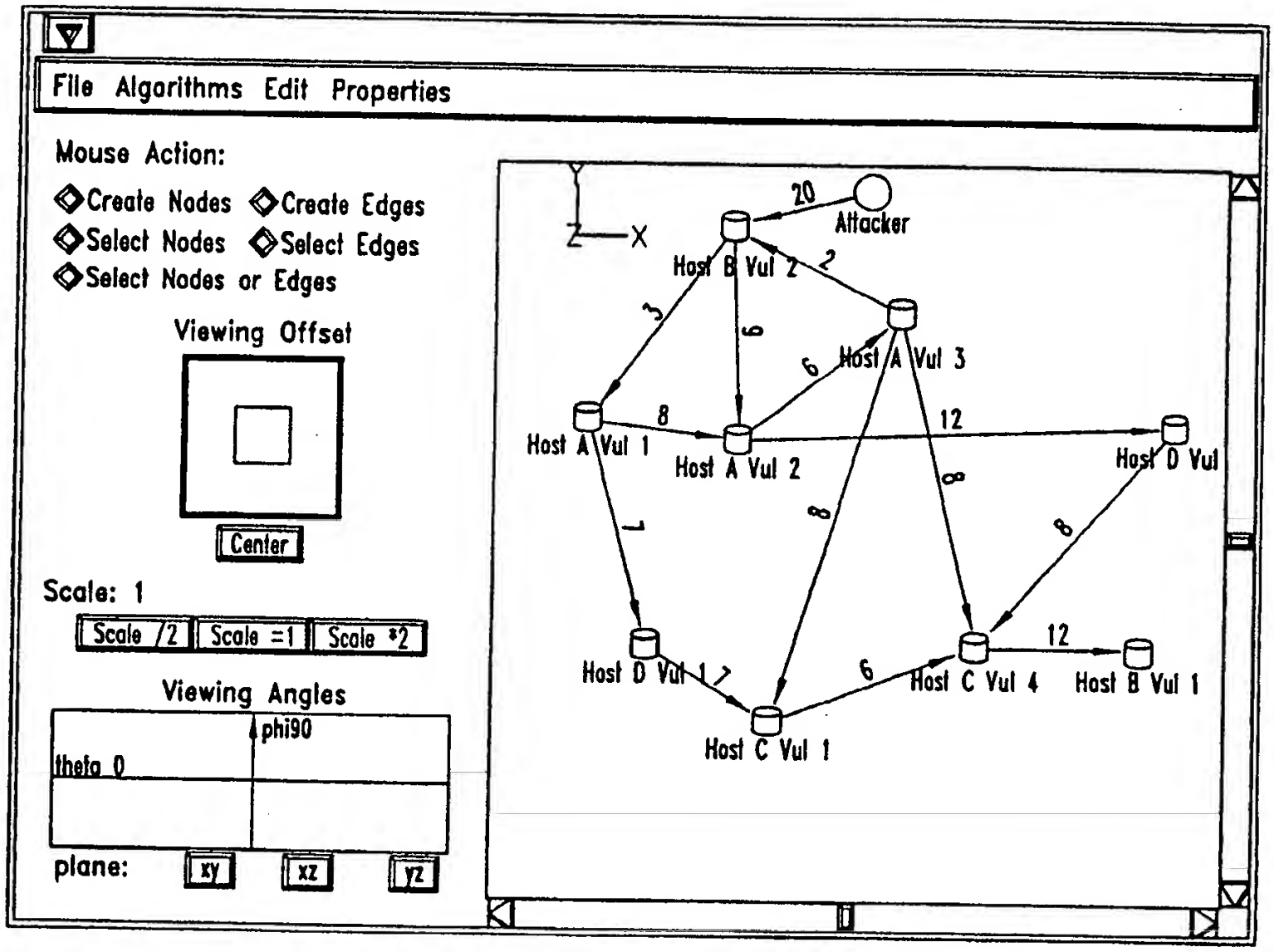

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
Attacker
20
Host B Vul 2
2
3
6
6
Host A Vul 3
8
12
Host A Vul 1
Host A Vul 2
Host D Vul
1
8
8
8
Host D Vul 1
7
6
12
Host C Vul 4
Host B Vul 1
Host C Vul 1

FIG. 33

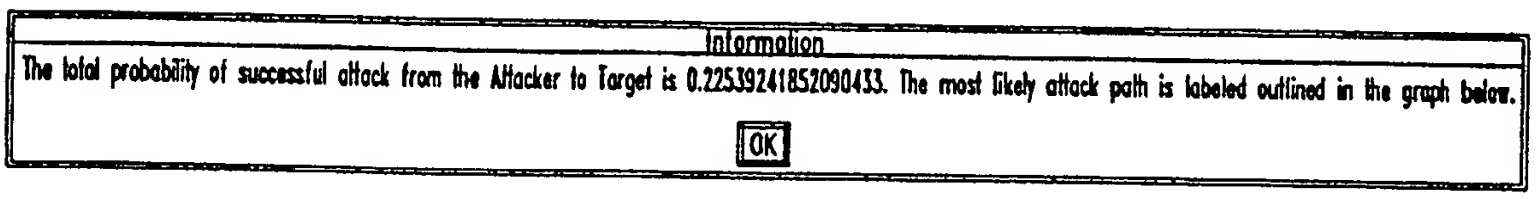

Information
The total probability of successful attack from the Attacker to Target is 0.22539241852090433. The most likely attack path is labeled outlined in the graph below.
OK

FIG. 34

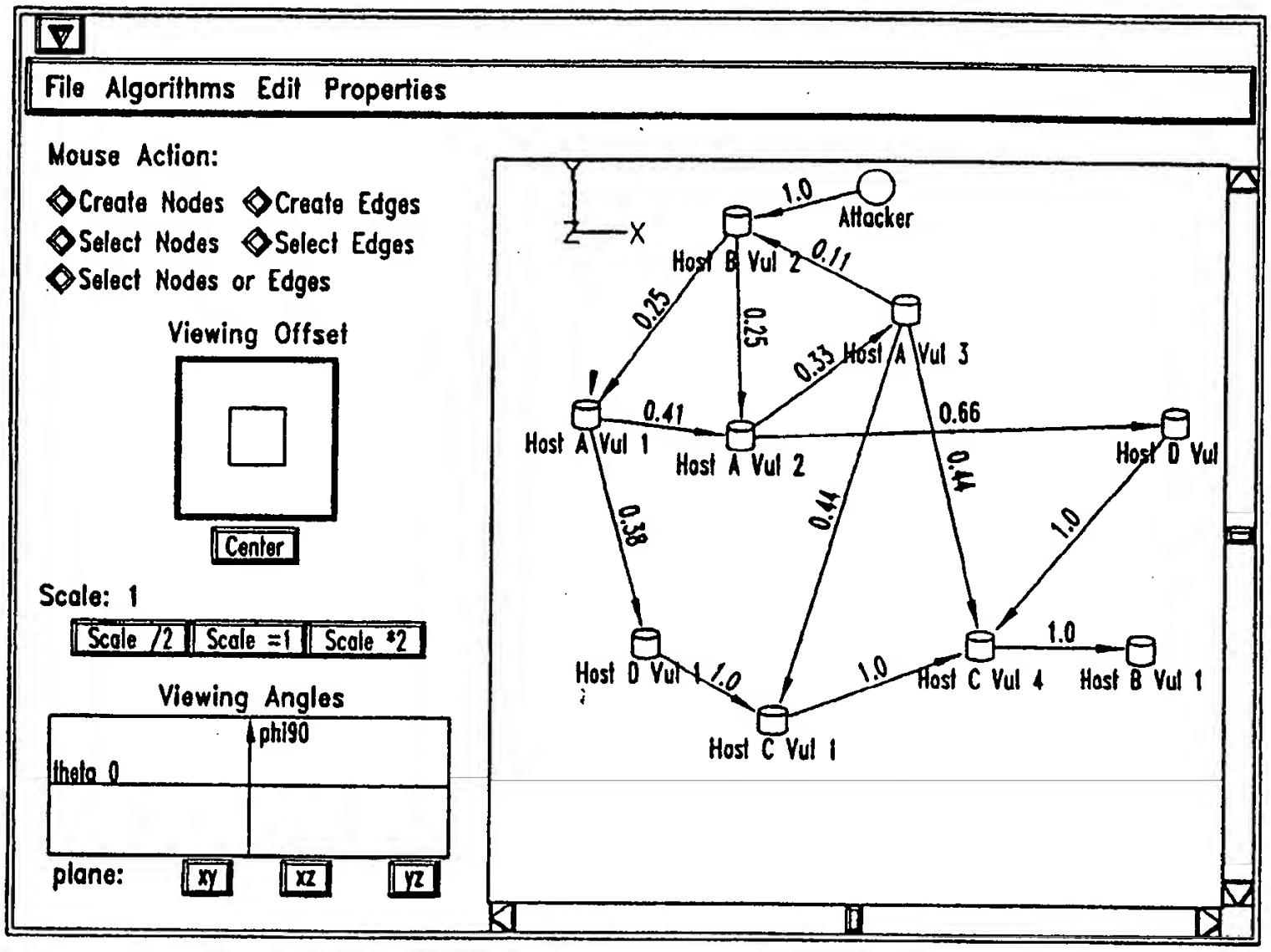

File Algorithms Edit Properties
Mouse Action:
Create Nodes
Create Edges
Select Nodes
Select Edges
Select Nodes or Edges
Viewing Offset
Center
Scale: 1
Scale /2
Scale =1
Scale *2
Viewing Angles
phi90
theta 0
plane:
xy
xz
yz
Attacker
1.0
Host B Vul 2
0.11
0.25
0.25
0.33
Host A Vul 3
0.41
0.66
Host A Vul 1
Host A Vul 2
Host D Vul
0.38
0.44
0.44
1.0
1.0
1.0
1.0
Host D Vul 1
Host C Vul 4
Host B Vul 1
Host C Vul 1

FIG. 35

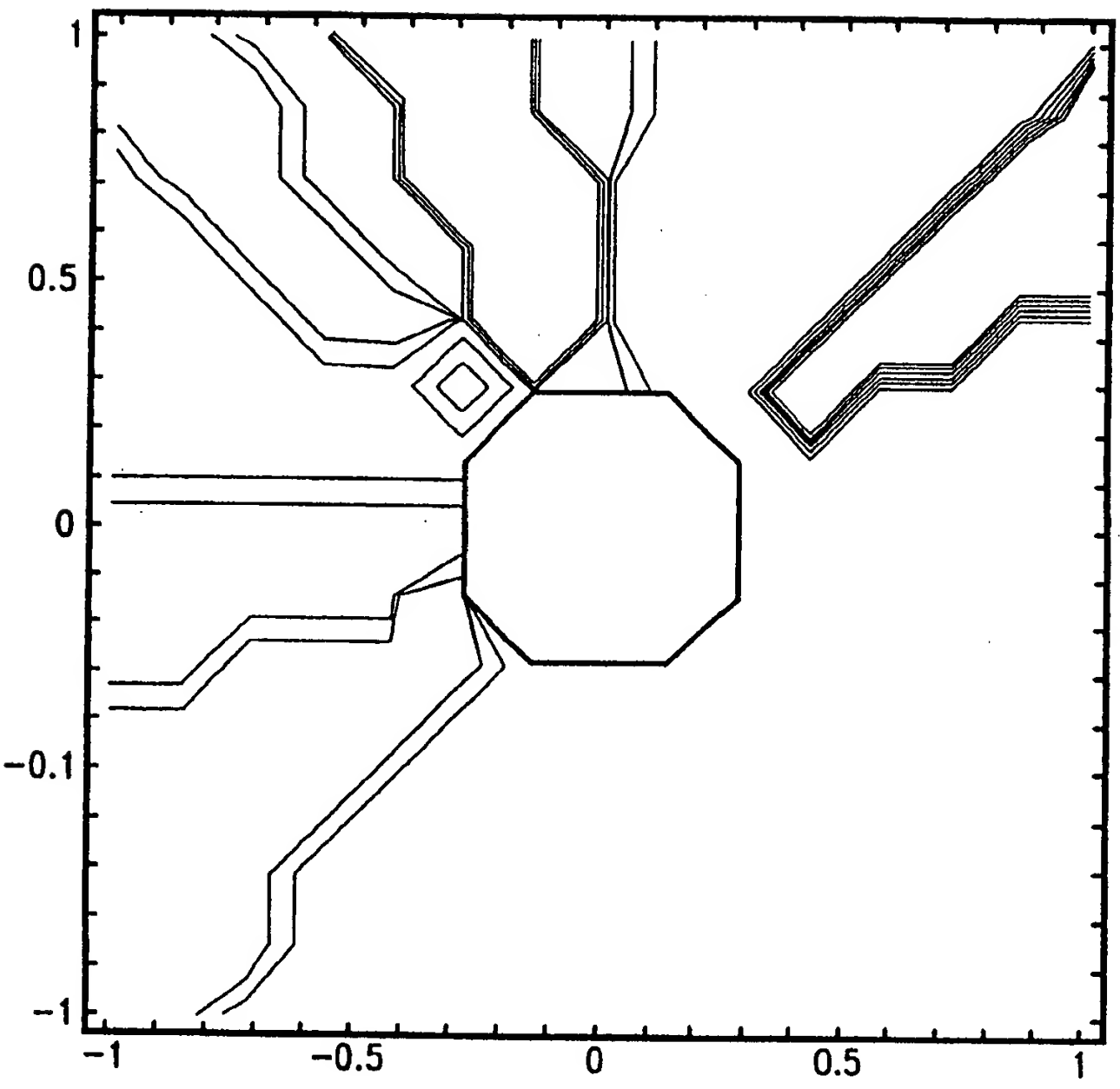


FIG. 36

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/12724

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 11/30
US CL : 713/201

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 713/201, 210; 714/1, 14; 345/440

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EAST
search terms: vulnerable, vulnerability, normalized, graph

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,892,903 A (KLAUS) 06 April 1999, All	1-18
Y	US 5,485,409 A (GUPTA et al) 16 January 1996, All	1-18
Y	US 5,684,957 A (KONDO et al) 04 November 1997, All	3, 4, 6, 10, 11, 16
A,E	US 6,088,804 A (HILL et al) 11 July 2000, All	
A,E	US 6,089,456 A (WALSH et al) 18 July 2000, ALL	

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date		
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 04 AUGUST 2000	Date of mailing of the international search report 24 AUG 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer ROBERT W BEAUSOLIEL JR Eugene Zogan Telephone No. (703) 308-7090

Form PCT/ISA/210 (second sheet) (July 1998)*